Design und Implementierung der FlexiTRUST-CA als Java Enterprise Application

Markus Winkler Lutz Feldgen

3. Dezember 2002

Diplomarbeit an der

Technischen Universität Darmstadt
Fachgebiet Theoretische Informatik
Kryptographie und Computeralgebra
Prof. Dr. Johannes Buchmann
Alexander Wiesmaier

Markus Winkler, Ludwigsplatz 8 a, 64283 Darmstadt, markus.winkler@gmxpro.de

Lutz Feldgen, Ludwigsplatz 8 a, 64283 Darmstadt, lfeldgen@gmx.de

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt 1. Dezember 2002

Markus Winkler Lutz Feldgen

Die in diesem Dokument erwähnten Soft- und Hardwarebezeichnungen sind in den meisten Fällen auch eingetragene Warenzeichen und unterliegen als solche den gesetzlichen Bestimmungen.

Vorwort

Da das Thema dieser Diplomarbeit zu umfassend für eine einzelne Diplomarbeit ist, wurde es in die Themenbereiche „Systemarchitektur“ und „Funktionalität“ aufgeteilt. Daher beschreibt das vorliegende Dokument zwei Diplomarbeiten, wobei das Thema Systemarchitektur Markus Winkler und das Thema Funktionalität Lutz Feldgen zuzuordnen ist. Diese Zuordnung ist jedoch nicht ausschliesslich zu verstehen, es handelt sich hierbei um Verantwortungsbereiche, deren Bearbeitung hauptsächlich durch den jeweils zugeordneten Diplomanden vorgenommen wurde.

Diese Aufteilung umfasst jedoch nicht sämtliche Aufgabengebiete, daher werden zur Kennzeichnung der jeweiligen Bearbeiter der einzelnen Kapitel und Abschnitte zusätzlich die Kürzel „MW“ für Markus Winkler und „LF“ für Lutz Feldgen angegeben.

Inhaltsverzeichnis

1 Einleitung **9**

1.1 Über dieses Handbuch 9

1.2 Struktur dieses Handbuches 10

1.3 Konventionen 10

1.4 Verwendete Werkzeuge 10

1.5 Zielumgebung 11

1.5.1 Hardware 11

1.5.2 Software 11

2 JWorkshop **12**

2.1 Beschreibung des JWorkshop 12

2.2 Umsetzung des JWorkshop 13

3 Die FlexiTRUST-CA **15**

3.1 Beschreibung einer CA 15

3.2 Funktionalitäten der FlexiTRUST-CA 16

3.2.1 Ausstellung eines X509v3 Zertifikats 16

3.2.2 Revokation eines Zertifikats 17

3.2.3 Einbringen eines Issuers 18

3.2.4 Einbringen eines Administrators 19

3.2.5 Zugriff auf gespeicherte Daten 19

3.3 Abbildung der CA auf den JWorkShop 20

3.3.1 Anträge 20
3.3.2 Speicherung von Daten 20
3.3.3 Funktionen zur Bearbeitung von Anträgen 20
3.3.4 Antragstellung 20
3.3.5 Antragsfertigstellung 20
3.4 Spezifikation der Anwendungsfälle 21
3.4.1 Anwendungsfälle der Domain Bench 21
3.4.2 Anwendungsfälle der Domain Entrance 22
3.4.3 Anwendungsfälle der Domain Exit 23
3.4.4 Anwendungsfälle der Domain Request 24
3.4.5 Anwendungsfälle der Domain Stock 24
3.5 Identifikation der Geschäftsklassen 27
3.5.1 Geschäftsklassen der Domain Bench 27
3.5.2 Geschäftsklassen der Domain Entrance 28
3.5.3 Geschäftsklassen der Domain Exit 29
3.5.4 Geschäftsklassen der Domain Request 30
3.5.5 Geschäftsklassen der Domain Stock 31

4 Systemarchitektur **34**

4.1 Architektur EIS-Tier 36
4.2 Architektur EJB-Tier 36
4.2.1 Bench 37
4.2.2 Entrance 38
4.2.3 Exit 39
4.2.4 Request 39
4.2.5 Stock 42
4.3 Architektur Web-Tier 47
4.4 Architektur Client-Tier 48
4.4.1 Anwendungsfälle des Pkcs7Server 48
4.4.2 Architektur Pkcs7Server 51

4.5 Grundlagen und Begründungen für die vorgestellte Architektur 55
4.5.1 Local-Interfaces der Entity-Beans 55
4.5.2 HA-JNDI und die Pkcs7Server-Session 56
4.5.3 EJBServiceLocator 57
4.5.4 Transaction Handling 57
4.5.5 Unittests 58
4.5.6 XDoclet 59
4.5.7 Value Objects 59

5 Umsetzung der fachlichen Anforderungen 61
5.1 Clustering 61
5.2 Logging 62
5.3 Schnittstellen zu externen Systemen 62
5.4 Konfiguration und Management 62

6 Datenmodell 63
6.1 Übersicht 63
6.2 Die Tabellen 63

7 Installation des Systems 66
7.1 Systemarchitektur 66
7.1.1 Die Ant-Build-Files 66
7.2 Datenbank Server 69
7.2.1 Systemvoraussetzungen 70
7.2.2 Installation des Datenbank Masters 70
7.2.3 Installation eines Replikanten 72
7.2.4 Update der Anwendung 72
7.2.5 Deinstallation 72
7.3 Enterprise Application Server 73
7.3.1 Systemvoraussetzungen 73
7.3.2 Vorbereitende Schritte 73

7.3.3 Installation der Anwendung 73
7.3.4 Start der FlexiTRUST-CA 76
7.3.5 Initialisierung der FlexiTRUST-CA 77
7.3.6 Herunterfahren der FlexiTRUST-CA 77
7.3.7 Update der Anwendung 77
7.3.8 Deinstallation 78
7.3.9 Distributionsversion der FlexiTRUST-CA 78

8 Betrieb des Systems **79**
8.1 Datenbank Server 79
8.2 Application Server 79
8.3 Pkcs7Server 80

9 Erweiterung des Systems **82**
9.1 Spezifikation der Anwendungsfälle und Klassen 82
9.1.1 Spezifikation der Anwendungsfälle 83
9.1.2 Identifikation der Geschäftsklassen 85
9.2 Systemarchitektur 87
9.2.1 Architektur EJB-Tier 88
9.2.2 Architektur Pkcs11Server 88
9.3 PKCS#11 Workbench 90
9.3.1 Die Klasse Pkcs11BenchEJB 90
9.3.2 Die Klasse Pkcs11Bench 95
9.3.3 Die Klasse Equipment 96
9.3.4 Die Klasse EntranceEJB 97
9.4 PKCS#11 Request 99
9.4.1 Die Klasse X509P11Request 99
9.4.2 Die Klasse X509P11RequestData 103
9.4.3 Die Klasse Request 104
9.4.4 Die Klasse RequestDAO 106

9.4.5 Die Klasse Types 113
9.5 PKCS#11 MBean-Service 113
9.5.1 Die Klasse Pkcs11CardAccess 114
9.5.2 Die Klasse Pkcs11SessionEJB 116
9.6 Buildfiles 118

10 Abschließende Bemerkungen **119**
10.1 Ausblick 119
10.1.1 Datenbank 119
10.1.2 Workbenches und Requests als Plug-In 120
10.1.3 CaStock 121
10.1.4 Benutzerschnittstelle 121
10.2 Fazit 122

Abbildungsverzeichnis

2.1 JWorkshop 13

3.1 Anwendungsfälle der Domain Bench 21
3.2 Anwendungsfälle der Domain Entrance 22
3.3 Anwendungsfälle der Domain Exit 23
3.4 Anwendungsfälle der Domain Request 24
3.5 Anwendungsfälle der Domain Stock 25
3.6 Geschäftsklassen der Domain Bench 28
3.7 Geschäftsklassen der Domain Entrance 29
3.8 Geschäftsklassen der Domain Exit 29
3.9 Geschäftsklassen der Domain Request 30
3.10 Geschäftsklassen der Domain Stock 32

4.1 Business Tier und EIS Tier 34
4.2 Struktur FlexiTRUST-CA 36
4.3 Enterprise JavaBeans Bench 37
4.4 Enterprise JavaBeans Entrance 38
4.5 Enterprise JavaBeans Exit 39
4.6 Enterprise JavaBeans Request 40
4.7 Enterprise JavaBeans Stock 43
4.8 Enterprise JavaBeans CaStock 46
4.9 JMX.Browser Agent View 47

4.10 Anwendungsfälle des Pkcs7Server Entrance 48
4.11 Anwendungsfälle des Pkcs7Server Exit 50
4.12 Anwendungsfälle des Pkcs7Server Session 51
4.13 MBean Service Pkcs7Entrance 52
4.14 MBean Service Pkcs7Exit 54
4.15 Enterprise JavaBeans Pkcs7Session 55
4.16 Sequenzdiagramm processRequest 56

6.1 Datenmodell 63

7.1 Systemarchitektur 67

8.1 MBean Detail Pkcs7Entrance 80
8.2 MBean Detail Pkcs7Exit 81

9.1 Anwendungsfälle der Domain Bench 83
9.2 Anwendungsfälle der Domain Request 84
9.3 Anwendungsfälle des Pkcs11Server CardAccess 84
9.4 Anwendungsfälle des Pkcs11Server Session 85
9.5 Geschäftsklassen der Domain Bench 86
9.6 Geschäftsklassen der Domain Request 87
9.7 Enterprise JavaBeans Bench 88
9.8 MBean Service Pkcs11CardAccess 89
9.9 Enterprise JavaBeans Pkcs11Session 90

Listings

7.1 ant.properties 74

7.2 ca.properties 75

9.1 domain.bench.Pkcs11BenchEJB 91

9.2 domain.bench.Pkcs11Bench 96

9.3 domain.bench.Equipment 96

9.4 ejbs.entrance.EntranceEJB 97

9.5 domain.request.X509P11Request 100

9.6 domain.request.data.X509P11RequestData 103

9.7 domain.request.Request 104

9.8 ejbs.request.RequestDAO 106

9.9 domain.request.Types 113

9.10 pkcs11server.cardaccess.Pkcs11CardAccess 114

9.11 pkcs11server.session.Pkcs11SessionEJB 116

Kapitel 1

Einleitung

MW+LF

1.1 Über dieses Handbuch

Das vorliegende Dokument beschreibt den Ansatz für die Umsetzung der Funktionalitäten der FlexiTRUST Certification Authority (FlexiTRUST-CA) auf DV-technischer Ebene.

Grundlage für die Entwicklung der FlexiTRUST-CA ist der ebenfalls im Rahmen dieser Diplomarbeit in J2EE neu entwickelte JWorkshop (JWS). Er definiert auf allgemeiner Ebene ein System für die Durchführung von Arbeitsabläufen und somit eine mögliche Architektur für Applikationen wie die hier vorgestellte CA oder andere Anwendungen. Weiterhin werden die Voraussetzungen für den Betrieb der FlexiTRUST-CA und die notwendigen Maßnahmen für die Installation der verschiedenen Komponenten erläutert.

Das Handbuch richtet sich an Entwickler, welche die CA erweitern oder einen anderen Teil von FlexiTRUST in der gleichen Architektur implementieren wollen. Hierfür werden grundlegende Kenntnisse in der Programmierung in J2EE [Inc02a] vorausgesetzt. Weiterhin ist es ein Installations- und Benutzerhandbuch für System-Administratoren, welche die CA installieren und administrieren. Hier werden werden Kenntnisse in UNIX- und/oder Windows-System-Administration und Administration von MySQL Datenbanken vorausgesetzt.

1.2 Struktur dieses Handbuches

Kapitel	Inhalt
1	Einleitung
2	JWorkshop
3	Die FlexiTRUST-CA
4	Systemarchitektur
5	Umsetzung der fachlichen Anforderungen
6	Datenmodell
7	Installation des Systems
8	Betrieb des Systems
9	Erweiterung des Systems
10	Abschließende Bemerkungen

1.3 Konventionen

Dieses Handbuch verwendet folgende Konventionen:

`Benutzereingabe`	In dieser Schriftart sind die vom Benutzer wörtlich einzugebenden Kommandos und Texte dargestellt.
`Ausgabe des Systems`	In dieser Schriftart werden Ausgaben des Systems dargestellt. Sie wird auch verwendet, um Dateiinhalte zu zeigen oder um im fortlaufenden Text einen exakten Dateinamen, ein Kommando, einen Pfadnamen oder ein Verzeichnis anzugeben.
`<Dateiname>`	In Beispielen, Kommandos oder fortlaufendem Text werden italics benutzt, um Platzhalter zu kennzeichnen, die im aktuellen Kontext vom Benutzer entsprechend einzusetzen sind.
`[Option]`	Beschreibung von Optionen sind fett und in eckigen Klammern angegeben.

Die im Handbuch angegebenen Kommandos entsprechen der UNIX-Syntax.

1.4 Verwendete Werkzeuge

Wesentliche Werkzeuge im Rahmen des Designs und der Entwicklung sind:

- Together/J Version 6.0 für die Geschäftsklassenmodellierung und als Front-End zum Debugging [Tog02],

- DbVisualizer 3.1 [Sof02] als Frontend für den Datenbankzugriff,
- Java 2 Platform Standard Edition 1.4.1_01 als Entwicklungsumgebung,
- AspectJ 1.1 [Org02] für die Generierung von Log-Statements im Source-Code,
- Emacs/JDE für die Source-Code-Erstellung und als Front-End für die Kompilierung und Versionsverwaltung,
- CVS für die Versionsverwaltung,
- Jakarta Ant 1.5 für die Übersetzungs- und Packaging Vorgänge,
- XDoclet für die Generierung der Interfaces und Deployment Deskriptoren der Enterprise-JavaBeans,
- EJTools JMX Browser als alternatives Web-Front-End [Sou02],
- JUnit [EG02] in Verbindung mit JUnitEE [Ros02] als Testumgebung.

1.5 Zielumgebung

Die CA soll auf geclusterten Windows- und Unixsystemen lauffähig und an einen Datenbankserver angeschlossen sein.

1.5.1 Hardware

Die CA ist als Plattformunabhängige Java-Applikation implementiert, daher können alle Systeme eingesetzt werden, für die ein Java-Runtime-Environment existiert.

1.5.2 Software

Die CA wurde mit Linux (Kernel 2.4.19, GLibc-2.2.5) und Windows2000/WindowsXP Betriebssystemen getestet. Weitere Softweareanforderungen sind der Application-Server JBoss in Version 3.0.3 [Gro02b] und die Entwicklungsumgebung Java 2 Platform Standard Edition (J2SE) in Version 1.4.1_01. Sollen die Quelltexte übersetzt werden, wird das Software Development Kit benötigt, für die Distributionsversion genügt das Java Runtime Environment [Inc02c].

Kapitel 2

JWorkshop

MW

Der JWorkshop ist die rudimentäre Implementierung einer Applikationsstruktur und der in dieser vorkommenden allgemeinen Arbeitsschritte. In diesem Kapitel wird der JWorkshop nur kurz beschrieben, da er Grundlage und nicht Thema dieser Diplomarbeit ist.

2.1 Beschreibung des JWorkshop

Der JWorkshop kann mit einer Werkstatt in der realen Welt verglichen werden. In einer Werkstatt gibt es eine Eingangstür, durch die Arbeiter in die Werkstatt kommen. Die Arbeiter wollen in der Werkstatt ein Werkstück, das sie mitbringen oder aus dem Lager holen, bearbeiten. Dafür gehen sie von der Eingangstür zu einer Arbeitsbank, die ein Gerät bereitstellt, das sie zur Bearbeitung des Werkstücks benötigen, beispielsweise eine Bohrmaschine. Ist ihre Arbeit an dieser Werkbank abgeschlossen, können sie mit ihrem Werkstück zu einer anderen Werkbank gehen, die ein anderes Gerät bereitstellt, beispielsweise eine Drehbank. Hier bearbeiten sie das Werkstück weiter. Sie können alle Werkbänke benutzen, die für die Bearbeitung ihres Werkstücks geeignet sind. Falls sie dafür zusätzliches Material benötigen, besorgen sie sich dieses aus dem Lager. Nicht verbrauchtes Material oder verwertbares Material, das bei der Bearbeitung anfällt und nicht dem Arbeiter gehört, kann im Lager abgelegt werden. Ist die Bearbeitung des Werkstücks abgeschlossen, gehen sie mit dem Werkstück zur Ausgangstür und verlassen die Werkstatt. Tritt bei einem der Vorgänge ein Fehler auf, müssen sie selbst entscheiden, wie mit diesem Fehler umgegangen werden soll. Sie tragen bei allen Vorgängen die Verantwortung für ihr Handeln und ihre Arbeitsergebnisse.

Dieses einfache Werkstatt-Modell diente als Grundlage für die Neuimplementierung des JWorkshop im Rahmen dieser Diplomarbeit.

2.2 Umsetzung des JWorkshop

Da in der Informatik die Sprache Englisch bevorzugt verwendet wird, sind die einzelnen Elemente der Beschreibung wie folgt übersetzt:

Worker bezeichnet einen Arbeiter, Entrance bezeichnet die Eingangstür, Workbench bezeichnet eine Werkbank, Stock bezeichnet das Lager und Exit bezeichnet die Ausgangstür.

Die Zusammenhänge sind in der folgenden Abbildung 2.1 graphisch dargestellt.

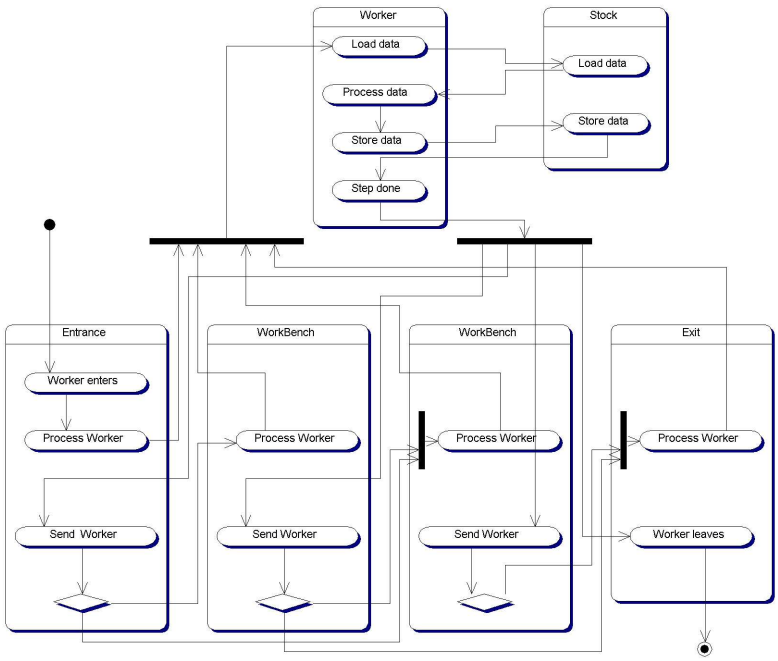

Abbildung 2.1: JWorkshop

Die Beschreibung des JWorkshop wird direkt in J2EE-Architektur umgesetzt:

Ein Worker stellt eine eigenständige Einheit dar und besitzt Eigenschaften, die ihn spezifizieren. In der J2EE-Architektur werden solche Einheiten als Entity-Beans umgesetzt. Diese sind für das Speichern, das Laden und die Aktualisierung ihrer eigenen Daten selbst verantwortlich und enthalten nur eingeschränkt darüber hinaus gehende Funktionalität. Hierfür spezifiziert J2EE Stateless-Session-Beans, die diese als unabhängige Services einer Entity-Bean zur Verfügung stellen. Da Exit, Entrance, Bench und Stock dem Worker solche Funktionalitäten zur Verfügung stellen, werden sie als

Stateless-Session-Beans realisiert [1].

Über abstrakte Klassen und Interfaces im JWS wird die Struktur für auf JWS aufsetzende Applikationen definiert.

[1] Für weitergehende Informationen über die verschiedenen Bean-Typen sei an dieser Stelle auf die Studienarbeit [Win02] verwiesen.

Kapitel 3

Die FlexiTRUST-CA

LF

3.1 Beschreibung einer CA

FlexiTRUST ist ein am Fachgebiet Theoretische Informatik der TU-Darmstadt entwickeltes Trustcenter. Ein Trustcenter bietet die Möglichkeit, die Authentizität, Integrität und Verbindlichkeit von digitalen Daten zu verifizieren. Kern der Verfahren, die dies ermöglichen, sind digitale Signaturen und Zertifikate, für deren Erstellung eine Certification-Authority (CA) zuständig ist.

Über einen Operator werden Anträge von einer Registration Authority (RA) zur Erstellung von Zertifikaten und Anträge auf deren Revokation an die CA herangetragen. Eine Revokation ist das Zuzückziehen eines Zertifikats. Da die Daten, die eine CA verarbeitet, äußerst sensibel sind, werden hohe Anforderungen an die Sicherheit und Richtigkeit der verarbeiteten Daten gestellt. Um die CA von vornherein vor eventuellen Angriffen über das Netzwerk zu schützen, wird sie entweder komplett vom Netzwerk des Trustcenters abgetrennt und der Datenaustausch geschieht über sichere Wege wie transportable Medien, oder sie ist nur für einen bestimmten Teil des Trustcenters zugänglich, der seinerseits besondere Schutzmaßnahmen aufweist.

Die CA verarbeitet Zertifikatsanträge, das heißt, sie vervollständigt vorbereitete Zertifikate und signiert sie anschliessend durch einen Aussteller. Zusätzlich speichert sie die Zertifikate für den internen Gebrauch. Auf Anfrage können Zertifikate zurückgezogen werden, wobei einem solchen Antrag eine Liste aller Revokationen für seinen weiteren Weg mitgegeben wird. Vollständig abgearbeitete Anträge werden an die Infrastructure Services (IS) übergeben.

Zur Inbetriebnahme der CA ist es erforderlich, einen initialen Aussteller und einen initialen Adminstrator einzubringen, bevor der erste Antrag gestellt wird. Ansonsten

wäre es nicht möglich, einen Antrag zu Verifizieren oder ein Zertifikat zu Signieren (vgl. Abschnitt 7.3.5 „Initialisierung der FlexiTRUST-CA“).

Zunächst wird im folgenden Abschnitt beschrieben, welche Funktionalitäten die FlexiTRUST-CA liefern soll. Aus diesen Funktionalitäten werden in den darauf folgenden Abschnitten Anwendungsfälle abgeleitet und die sie realisierenden Klassen definiert. Als Grundlage dient hierfür die bestehende Implementierung der FlexiTRUST-CA [Wie01].

3.2 Funktionalitäten der FlexiTRUST-CA

Die Hauptaufgabe der FlexiTRUST-CA ist die Austellung von Zertifikaten und deren Revokation. Daraus ergeben sich die im folgenden beschriebenen Grundfunktionalitäten, die über Anträge in Anspruch genommen werden können.

3.2.1 Ausstellung eines X509v3 Zertifikats

Ausstellungsantrag

Ein solcher Antrag beinhaltet ein von der RA vorbereitetes und teilweise befülltes Zertifikat. Dieses enthält unter anderem den Namen des zukünftigen Besitzers des Zertifikats[1], den Namen des Ausstellers[2], den Zeitraum für die Gültigkeit des Zertifikats und den öffentlichen Schlüssel des Besitzers. Zusätzlich zu dem Zertifikat wird in dem Antrag ein Paßwort für die Authorisation zu einer eventuellen Revokation des Zertifikats abgelegt.

Empfang und Verifikation des Antrags

Der Antrag muß digital signiert der CA übergeben werden. Die CA verifiziert den Antrag, um dessen Echtheit sicherzustellen. Die Operator-Aufgaben Signierung und Übergabe an die CA übernimmt ein Administrator. Mit dessen privaten Schlüssel wird der Antrag unterschrieben. Für die effizientere Übergabe an die CA können mehrere solcher Anträge zusammengefasst werden, das Gesamtpaket wird wiederum mit dem privaten Schlüssel des Administrators unterschrieben. Es können mehrere Administratoren existieren, allerdings muß jeder Administrator, der in der RA einen Antrag oder ein Gesamtpaket unterschreibt, in der CA bekannt sein, da für die Verifikation der Anträge und Gesamtpakete der öffentliche Schlüssel des unterschreibenden Administrators benötigt wird.

[1] Der Besitzer eines Zertifikats wird in FlexiTRUST User genannt.
[2] Der Aussteller eines Zertifikats wird in einer Public-Key-Infrastruktur(PKI) Issuer genannt.

Dieser Vorgang ist bei allen Anträgen, die in die CA gebracht werden, gleich.

Vervollständigung und Signierung des Zertifikats

Der nächste Schritt ist die Vervollständigung der Daten des Zertifikats. Hier wird der Signaturalgorithmus eingetragen und eine Seriennummer für das Zertifikat generiert. Die Seriennummer und der Issuer zusammen liefern eine eindeutige Identifikation des Zertifikats. Innerhalb der FlexiTRUST-CA ist hierfür allerdings bereits die Seriennummer ausreichend. Anschließend wird das Zertifikat mit dem privaten Schlüssel des Issuers signiert und in dem Antrag gespeichert.

Persistente Speicherung

Das Zertifikat wird zusätzlich in der CA persistent gespeichert. Hierfür wird zunächst der Besitzer angelegt, sofern er noch nicht in der CA bekannt ist. Danach wird es Zertifikat mit seinem Revokationspaßwort gespeichert und mit Antrag, Besitzer und Issuer verküpft. Dadurch ist es möglich, jederzeit festzustellen, mit welchem Antrag ein Zertifikat in die CA gebracht wurde, wem es gehört und wer es ausgestellt hat.

Genauso wie mehrere Administratoren unterstützt werden, können auch mehrere Issuer in der CA existieren.

Weiterhin wird der private Schlüssel des Besitzers mit dem zugehörigen Paßwort gespeichert und mit dem Zertifikat verknüpft. Die Bearbeitung des Antrags ist hiermit abgeschlossen.

Signierung und Ausgabe des Antrags

Zur Übergabe an die auf die CA folgende Instanz IS erfolgt die Signierung des Antrags mit dem privaten Schlüssel eines Administrators. Dieser Administrator muß nicht derselbe sein, der den Antrag in die CA gebracht hat, er muß mit seinem öffentlichen Schlüssel lediglich für die IS verfügbar sein. Falls innerhalb eines festgelegten Zeitraums mehrere Anträge fertig bearbeitet wurden, werden diese Anträge zu einem Gesamtpaket zusammengefasst und dieses wiederum signiert und letztlich an die IS übergeben.

Dieser Vorgang ist bei allen Anträgen, die aus der CA gebracht werden, gleich.

3.2.2 Revokation eines Zertifikats

In bestimmten Fällen ist es notwendig, ein Zertifikat für ungültig zu erklären. Diesen Vorgang nennt man Revokation, er beinhaltet außerdem die Erstellung einer Liste von

bisherigen Revokationen.

Revokationsantrag

Ein Revokationsantrag beinhaltet den Namen des Issuers, das Revokationspaßwort und einen Revokationseintrag, der aus der Seriennummer des zu revozierenden Zertifikats und dem Zeitpunkt der Revokation zusammengesetzt ist. Die Einbringung des Antrags erfolgt analog zu Abschnitt 3.2.1 „Ausstellung eines X509v3 Zertifikats“.

Revokation

Zunächst wird überprüft, ob das zu revozierende Zertifikat in der CA existiert. Weiterhin wird das übergebene Paßwort mit dem gespeicherten Paßwort des Zertifikats verglichen und die Revokation zurückgewiesen, wenn sie nicht übereinstimmen.

Ist das Paßwort korrekt, wird überprüft, ob das Zertifikat bereits zurückgezogen wurde. Ist dies der Fall, wird die Revokation selbst abgebrochen, der Antrag aber weiter verarbeitet um die Revokationsliste zu liefern. Ist das Zertifikat noch gültig, wird der Revokationseintrag mit dem Verfallsdatum des Zertifikats persistent gespeichert und mit dem zurückgezogenen Zertifikat verknüpft.

Revokationsliste

Anschließend wird die Liste von Revokationseinträgen erzeugt. Diese Liste beinhaltet unter anderem den Issuer des Revokationsantrags, das Erstellungsdatum und die Revokationseinträge. In diese Liste nimmt die FlexiTRUST-CA nur Revokationseinträge auf, wenn die Verfallsdaten der zurückgezogenen Zertifikate noch nicht überschritten sind. Es kann allerdings auch eine vollständige Liste mit allen Revokationseinträgen erstellt werden.

Anschliessend wird der Antrag mit der erstellten Revokationsliste analog zu Abschnitt 3.2.1 „Ausstellung eines X509v3 Zertifikats“ an die IS übergeben.

3.2.3 Einbringen eines Issuers

Die FlexiTRUST-CA unterstützt mehrere Issuer, die über Issueranträge in die CA eingebracht werden.

Issuerantrag

Dieser Antrag wird vollständig befüllt in die CA eingebracht. Er enthält unter anderem den Namen des neuen Issuers, dessen Zertifikat, die Seriennummer des Zertifikats, das Paßwort für eine eventuelle Revokation des Zertifikats, den privaten Schlüssel des Issuers und das Paßwort für den Zugriff darauf. Zusätzlich enthält der Antrag noch eine Kette von Zertifikaten. Dieser Kette kann rekursiv entnommen werden, wer das jeweilige Zertifikat ausgestellt hat und letztendlich verantwortlich für die Ausstellung des einzubringenden Issuerzertifikats und damit für die Vertrauenswürdigkeit des neuen Issuers verantwortlich ist.

Die Einbringung des Antrags erfolgt analog zu Abschnitt 3.2.1 „Ausstellung eines X509v3 Zertifikats“[3].

Persistente Speicherung

Der neue Issuer wird mit allen Daten persistent gespeichert. Hierbei wird sichergestellt, daß nicht bereits ein anderer Issuer mit demselben Namen eingetragen wurde.

Der Antrag wird nun ohne die persönlichen Daten des Issuers analog zu Abschnitt 3.2.1 „Ausstellung eines X509v3 Zertifikats“ an die IS übergeben.

3.2.4 Einbringen eines Administrators

Die CA unterstützt mehrere Administratoren, die über Anträge in die CA eingebracht werden können.

Das Einbringen eines neuen Administrators erfolgt analog zu Abschnitt 3.2.3 „Einbringen eines neuen Issuers“[4], es gibt nur einen Unterschied zu einem Issuerantrag: Der Administratorantrag enthält anstelle der Zertifikatskette den Namen des Issuers seines Zertifikats.

3.2.5 Zugriff auf gespeicherte Daten

Die CA bietet weitere Möglichkeiten, auf ihre gespeicherten Daten zuzugreifen. Beispiele hierfür sind das Abrufen einer Zertifikatskette eines beliebigen Issuer-, Admin-

[3] Dies ist beim momentanen Entwicklungsstand noch nicht gegeben, da keine Möglichkeit der Verifikation des initialen Issuers besteht. Um das gewünschte Verfahren zu realisieren, müsste bei Einbringung eines Issuers zwischen initialem und zusätzlichem Issuer unterschieden werden (siehe Abschnitt 10.1.3 „CaStock“).

[4] Dies ist beim momentanen Entwicklungsstand noch nicht gegeben, da keine Möglichkeit der Verifikation des initialen Administrators besteht. Um das gewünschte Verfahren zu realisieren, müsste bei Einbringung eines Administrators zwischen initialem und zusätzlichen Administrator unterschieden werden (siehe Abschnitt 10.1.3 „CaStock“).

istrator- oder Userzertifikats, das Abrufen des aktuellen Status eines momentan bearbeiteten Antrags, das Abrufen der vollständigen Revokationsliste, das Abrufen aller Zertifikate eines bestimmten Users und das Abrufen aller momentan gültigen Zertifikate eines bestimmten Users.

3.3 Abbildung der CA auf den JWorkShop

Um mit Hilfe des JWS die CA entwickeln zu können, muß die CA auf die Struktur des JWS abgebildet werden.

3.3.1 Anträge

Ein Antrag wird auf einen JWS-Worker abgebildet und Request genannt. Für die verschiedenen Anträge existieren spezialisierte Requests. Beispiele für Spezialisierungen sind Zertifikatsanträge, Revokationsanträge und Anträge auf eine neue Revokationsliste.

3.3.2 Speicherung von Daten

Das Speichern und Abrufen von allgemeinen persistenten Daten wird auf den JWS-Stock abgebildet. Der so innerhalb der CA erweiterte Stock wird CaStock genannt.

3.3.3 Funktionen zur Bearbeitung von Anträgen

Sämtliche Funktionen zur Bearbeitung von Anträgen werden auf JWS-Workbenches abgebildet. Für die verschiedenen Arten von Funktionen existieren spezialisierte Workbenches, ein Beispiel hierfür ist die CryptoBench für kryptographische Funktionen wie Verifikation und Signierung von Daten.

3.3.4 Antragstellung

Ein Operator trägt Anträge der RA in Dateiform an die CA heran. Sie werden von einem Leseprozeß, der als Entrance abgebildet wird, eingelesen.

3.3.5 Antragsfertigstellung

Ist ein Antrag vollständig bearbeitet, wird er von einem Schreibprozeß, der als Exit abgebildet wird, in Dateiform abgelegt und so der IS zugänglich gemacht.

3.4 Spezifikation der Anwendungsfälle

In diesem Abschnitt werden aus den in Abschnitt 3.2 „Funktionalitäten der FlexiTRUST-CA“ beschriebenen Funktionalitäten einzelne Anwendungsfälle abgeleitet und deren Strukturen im System dokumentiert. Diese Anwendungsfälle sind nach der in Abschnitt 3.3 „Abbildung der CA auf den JWorkshop“ ermittelten Struktur gegliedert und berücksichtigen bereits Implementierungsaspekte.

Die Darstellung erfolgt in UML-Notation.

3.4.1 Anwendungsfälle der Domain Bench

Die nachfolgende Abbildung 3.1 zeigt alle Anwendungsfälle des Bereichs `Bench` und ihre Beziehungen untereinander.

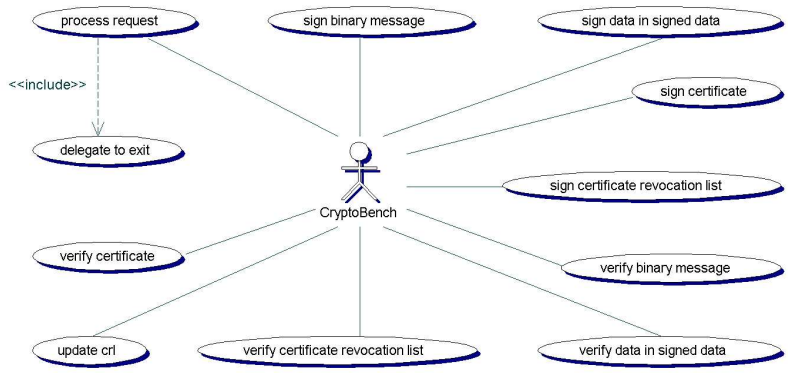

Abbildung 3.1: Anwendungsfälle der Domain Bench

- `process request`: Im Anwendungsfall findet die Abarbeitung eines oder mehrerer Arbeitsschritte des Requests auf der Bench statt. Hier wird eine lokale Instanz des Requests erzeugt und dessen Methode `work` aufgerufen. Die Bench stellt ihm hierbei ihre spezifischen Funktionalitäten zur Verfügung.
- `delegate to exit`: Nachdem ein Request alle auf der Bench möglichen und durchzuführende Arbeitsschritte abgeschlossen hat, wird er von dieser an die von ihm spezifizierte nächste Bench weitergegeben.
- `sign binary message`,
  `sign data in signed data`,
  `sign certificate` und

`sign certificate revocation list`: Die CryptoBench stellt den Requests Methoden zur Signierung unterschiedlicher Objekte zur Verfügung.

- `verify binary message`,
  `verify data in signed data`,
  `verify certificate`,
  `verify certificate revocation list`: Die CryptoBench stellt den Requests Methoden zur Verifikation unterschiedlicher Objekte zur Verfügung.
- `update crl`: Bietet die Möglichkeit, Revokationslisten zu aktualisieren.

3.4.2 Anwendungsfälle der Domain Entrance

Die nachfolgende Abbildung 3.2 zeigt alle Anwendungsfälle des Bereichs `Entrance` und ihre Beziehungen untereinander.

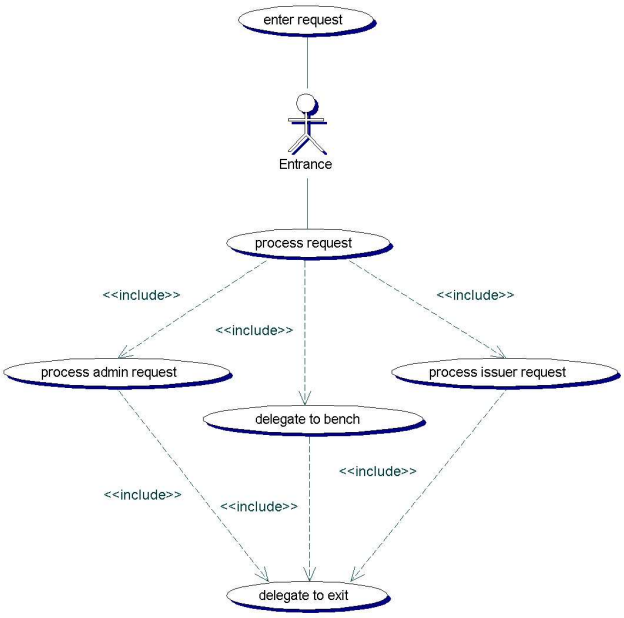

Abbildung 3.2: Anwendungsfälle der Domain Entrance

- `enter request`: Soll ein neuer Request von der CA verarbeitet werden, wird er, initiiert durch den Pkcs7EntranceService (vgl. Abschnitt 4.4.1 „Anwendungsfälle des Pkcs7Server), persistent in die Datenbank eingebracht. An diesen Fall schließt sich der Anwendungsfall `process request` an.

- `process request`: Der Anwendungsfall beschreibt die erstmalige Bearbeitung des Requests innerhalb des EJB-Tiers der FlexiTRUST-CA. Je nach Typ des Requests werden unterschiedliche Verarbeitungsschritte durchgeführt. Alle Typen werden von spezialisierten und vom Request definierten Workbenches verarbeitet.
- `process admin request`:
  `process issuer request`: Dies sind die Spezialisierungen des Anwendungsfalls `process request` für Admin- und Issuer-Request.

3.4.3 Anwendungsfälle der Domain Exit

Die nachfolgende Abbildung 3.3 zeigt alle Anwendungsfälle des Bereichs `Exit` und ihre Beziehungen untereinander.

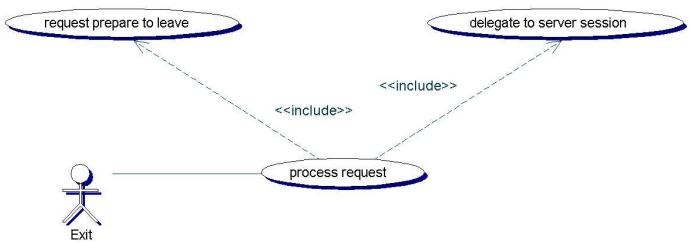

Abbildung 3.3: Anwendungsfälle der Domain Exit

- `process request`: Dieser Anwendungsfall der `ExitEJB` beinhaltet abschließende Verarbeitungsschritte des Requests. Im einzelnen werden hier Requests auf das Verlassen des EJB-Tiers vorbereitet und schließlich an den Pkcs7ExitService geleitet. Er untergliedert sich daher in die Fälle `request prepare to leave` und `delegate to server session`.
- `request prepare to leave`: In diesem Anwendungsfall wird der Request angewiesen, abschließende Schritte durchzuführen. Bei den zur Zeit implementierten Requests ist diese Methode leer und stellt daher Erweiterungsmöglichkeiten für zukünftig implementierte Typen dar.
- `delegate to server session`: In diesem Moment verläßt der Request das EJB-Tier der FlexiTRUST-CA. Er wird über die `Pkcs7ServerSessionEJB` an den Pkcs7ExitService geleitet (vgl. Abschnitt 4.4.1 „Anwendungsfälle des Pkcs7Server“).

3.4.4 Anwendungsfälle der Domain Request

Die nachfolgende Abbildung 3.4 zeigt alle Anwendungsfälle des Bereichs `Request` und ihre Beziehungen untereinander.

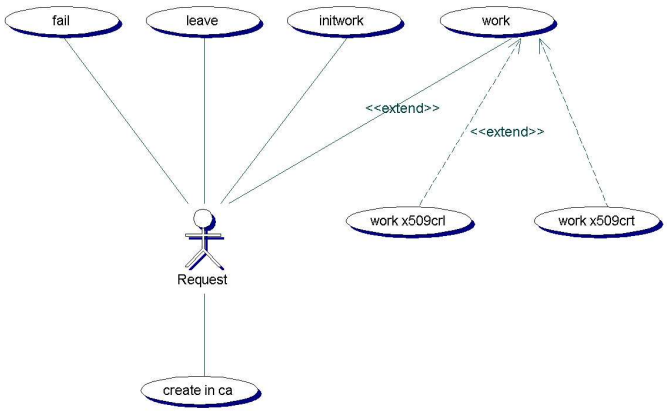

Abbildung 3.4: Anwendungsfälle der Domain Request

- `create in ca`: Der Request betritt die FlexiTRUST-CA und wird persistent mit seinen initialen Daten in die Datenbank eingebracht.
- `initwork`: Dieser Fall initialisiert den Request für die Bearbeitung.
- `work`: Dies ist die Implementierung der Arbeitsschritte der Requests. Jeder Typ von Request besitzt eine Spezialisierung dieser Methode.
- `work x509crl`,
  `work x509crt`: Dies sind Spezialisierungen des Anwendungsfalls `work`. Weitere Spezialisierungen werden hier hinzugefügt.
- `leave`: Dieser Fall bereitet den Request auf das Verlassen des EJB-Tiers der FlexiTRUST-CA vor.
- `fail`: Tritt im Workflowverlauf eines Requests ein Fehler auf, der zum Abbruch der Bearbeitung führt, wird er in der Datenbank mit `failed` gekennzeichnet.

3.4.5 Anwendungsfälle der Domain Stock

Die nachfolgende Abbildung 3.5 zeigt alle Anwendungsfälle des Bereichs `Stock` und ihre Beziehungen untereinander.

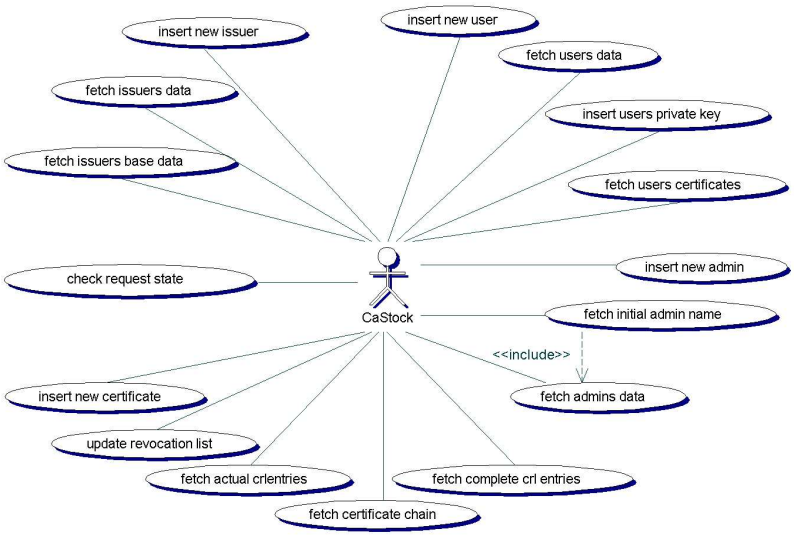

Abbildung 3.5: Anwendungsfälle der Domain Stock

Anwendungsfälle Bereich `Request`:

- `check request state`: Wird der aktuelle Status eines Requests benötigt, liefert der Stock diesen aus der Datenbank zurück.

Anwendungsfälle Bereich `Issuer`:

- `insert new issuer`: Soll ein neuer Issuer eingefügt werden, darf dieser Issuer nicht bereits eingetragen worden sein und muß eindeutig im System zu identifizieren sein. Beides wird von der Datenbank sichergestellt.
- `fetch issuers data`: Momentan existiert noch kein Anwendungsfall, der sämtliche Daten eines Issuers benötigt, die Funktionalität liefert der Stock für zukünftige Anwendungsfälle
- `fetch issuers base data`: Zur Signierung und Verifikation werden die Basisdaten des Issuers benötigt.

Anwendungsfälle Bereich `Admin`:

- `insert new admin`: Soll ein neuer Administrator eingefügt werden, darf dieser nicht bereits eingetragen worden sein und muß eindeutig im System zu identifizieren sein. Beides wird von der Datenbank sichergestellt.

- `fetch initial admin name`: Zur Verifikation eines eingehenden und zum Signieren eines ausgehenden Request-Pakets werden Daten eines Administrators benötigt, der mit seinem öffentlichen Schlüssel in der folgenden Instanz verfügbar sein muß. Momentan ist dies der initial eingebrachte Administrator, dessen Name über den Stock abgefragt wird.
- `fetch admins data`: Die Daten des Administrators werden zur Signierung oder Verifikation eines Request-Pakets benötigt.

Anwendungsfälle Bereich `User`:

- `insert new user`: Wird ein Zertifikatsantrag gestellt, muß der zukünftige Besitzer des Zertifikats in die Datenbank eingebracht und ein Primärschlüssel erstellt werden. Ist dieser User bereits bekannt, wird dies vom Stock erkannt und der gefundene Primärschlüssel zurückgegeben.
- `insert users private key`: Wird ein neues Zertifikat ausgestellt, kann der zu ihm passende Private-Key gespeichert werden. Zugriff auf diesen Private-Key darf nur erfolgen, wenn das richtige Paßwort des Schlüssels angegeben wurde.
- `fetch users data`: Dieser Anwendungsfall tritt nur dann auf, wenn getestet werden soll, ob die Einbringung eines Users erfolgreich war.
- `fetch users certificates`: Wenn nicht bekannt ist, welches der Zertifikate eines Users gebraucht wird, können alle Zertifikate eines Users vom Stock angefordert werden, um anschließend das entsprechende Zertifikat auszuwählen.

Anwendungsfälle Bereich `Zertifikate`:

- `insert new certificate`: Jedes von der CA erstellte Zertifikat muß gespeichert werden. Hierbei wird überprüft, ob das Zertifikat anhand seiner Seriennummer eindeutig identifiziert werden kann. Desweiteren muß sichergestellt sein, daß der Antragsteller im System bekannt ist. Zusätzlich wird überprüft, ob der Besitzer des Zertifikats im System bekannt ist oder ob er in die Datenbank eingefügt werden muß.
- `update revocation list`: Verliert ein Zertifikat seine Gültigkeit, muß dies der CA respektive dem Stock angezeigt werden. Bei einem solchen Revokationsantrag wird überprüft, ob das zurückzuziehende Zertifikat im Datenbestand vorhanden ist und ob es nicht bereits zurückgezogen wurde.
- `fetch actual crlentries`: Es muß möglich sein, eine Liste mit den aktuellen Revokationen anzufordern. Der Stock liefert hier eine reduzierte Liste mit allen

Revokationen zu Zertifikaten, die zum Zeitpunkt der Erstellung der Liste gültig sind.

- `fetch complete crlentries`: In manchen Fällen wird eine Liste aller Revokationen benötigt, diese kann vom Stock angefordert werden.
- `fetch certificate chain`: Wenn zu einem Zertifikat die Zertifikatskette der Issuer angefordert wird, liefert der Stock eine Liste von Zertifikaten der jeweiligen Issuer bis hin zu dem Issuer, der dieses Zertifikat in die CA gebracht hat.

3.5 Identifikation der Geschäftsklassen

Die mit Hilfe der Anwendungsfälle erkannten Geschäftsklassen werden in den folgenden Klassendiagrammen dargestellt. Die Darstellung erfolgt in UML-Notation. Die Menge aller Geschäftsklassen bildet die Grundlage des Design-Modells, d.h. die Darstellung entspricht nicht mehr der Analysesicht und berücksichtigt bereits Implementierungsaspekte (vgl. hierzu [Win02]).

3.5.1 Geschäftsklassen der Domain Bench

Im Paket `Bench` sind die zentralen Klassen der Workbenches mit den notwendigen Funktionalitäten zur Requestbearbeitung zusammengefasst.

Die in Abbildung 3.6 dargestellten Klassen werden in den nachfolgenden Abschnitten beschrieben. Im Rahmen einer Erweiterung der CA sind die Geschäftsklassen neuer Workbenches innerhalb dieses Packages zu implementieren (vgl. hierzu Kapitel 9 „Erweiterung des Systems“).

Klasse CryptoBench

Die Klasse `CryptoBench` stellt grundlegende kryptographische Funktionalitäten wie z.B. Verifizieren und Signieren von Byte-Arrays, SignedData-Objekten und X509Certificate-Objekten zur Verfügung. Weiterhin werden Methoden für die Generierung von KeyPairs bereitgestellt. Diese Funktionalitäten können direkt von den Requests in Anspruch genommen werden.

Klasse Equipment

Die Klasse `Equipment` beinhaltet eine Auflistung aller in der CA implementierten Benches. Die in ihr definierten Konstanten werden von den Requests und Workbenches zur Identifizierung der Benchtypen benötigt

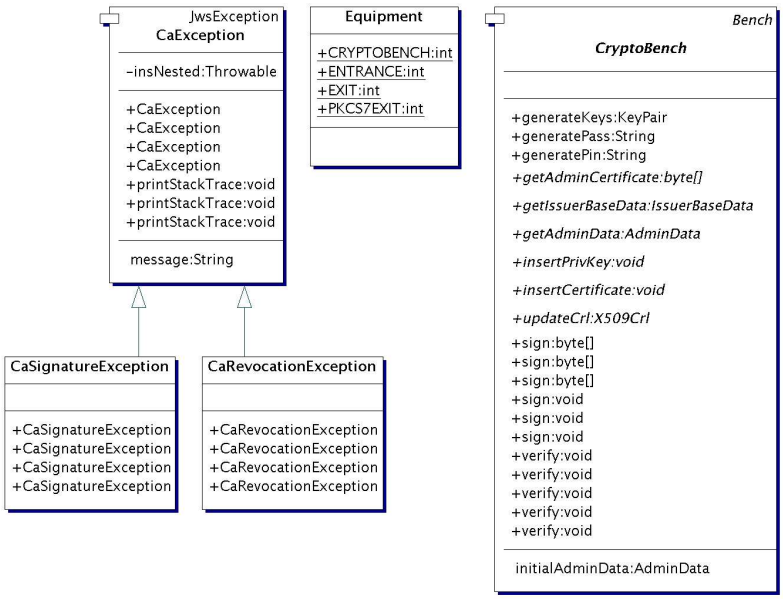

Abbildung 3.6: Geschäftsklassen der Domain Bench

Klasse CaException

Die Klasse `CaException` beschreibt einen allgemeinen Fehler, der innerhalb des Workflows der CA aufgetreten ist.

Klasse CaRevocationException

Die Klasse `CaRevocationException` beschreibt die Fehler, die bei der Revokation eines Zertifikats aufgetreten können. Sie ist eine Spezialisierung der Klasse `CaException`.

Klasse CaSignatureException

Die Klasse `CaSignatureException` beschreibt die Fehler, die bei der Verifikation einer Signatur aufgetreten können. Sie ist eine Spezialisierung der Klasse `CaException`.

3.5.2 Geschäftsklassen der Domain Entrance

Im Paket `Entrance` sind die zentralen Klassen des Entrance der CA zusammengefasst.

Die in Abbildung 3.7 dargestellte Klasse wird im nachfolgenden Abschnitt beschrieben.

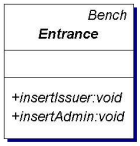

Abbildung 3.7: Geschäftsklassen der Domain Entrance

Klasse Entrance

Der `Entrance` stellt innerhalb des Workflow-Konzeptes der implementierten Flexi-TRUST-CA den Eingang in das EJB-Tier dar. Sie definiert direkt auf der `EntranceEJB` bereitzustellende Funktionalitäten. Im momentanen Entwicklungsstand sind dies das Einfügen von Issuern und Administratoren.

3.5.3 Geschäftsklassen der Domain Exit

Im Paket `Exit` sind die zentralen Klassen des Exit zusammengefasst.

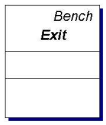

Abbildung 3.8: Geschäftsklassen der Domain Exit

Die in Abbildung 3.8 dargestellte Klasse wird in dem nachfolgenden Abschnitt beschrieben.

Klasse Exit

Der `Exit` stellt innerhalb des EJB-Tiers den Ausgang dar. Alle Requests werden vom ihm entgegengenommen und an den MBean-Service `Pkcs7Exit` übergeben (vgl. Abschnitt 4.4.2 „Architektur Pkcs7Server“).

3.5.4 Geschäftsklassen der Domain Request

Im Paket `Request` sind die zentralen Klassen der Requests zusammengefasst.

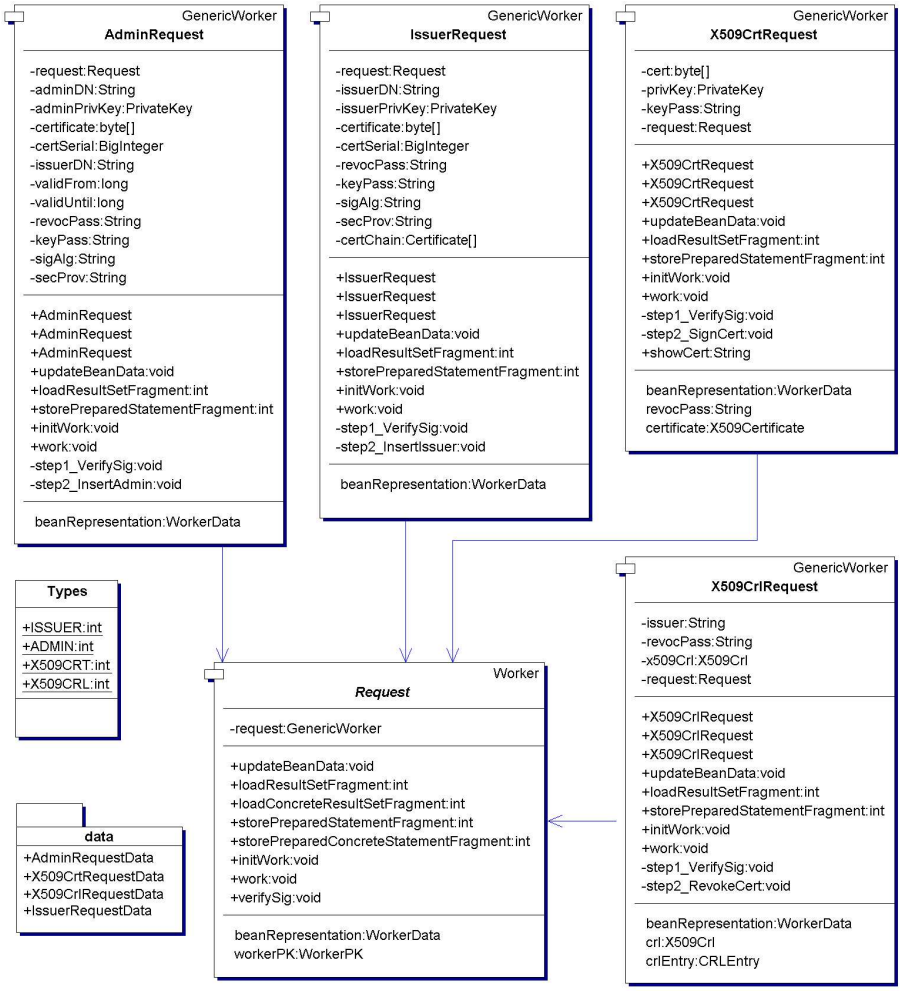

Abbildung 3.9: Geschäftsklassen der Domain Request

Die in Abbildung 3.9 dargestellten Klassen werden in den nachfolgenden Abschnitten beschrieben. Im Rahmen einer Erweiterung der CA sind die Geschäftsklassen neuer Requests innerhalb dieses Packages zu implementieren (vlg. hierzu Kapitel 9 „Erweiterung des Systems“).

Request

Die Klasse `Request` beschreibt einen Request innerhalb der FlexiTRUST-CA. Neben den beschreibenden Attributen beinhaltet sie einen Verweis auf einen der spezialisierten Requests. Der `Request` bietet unterstützende Funktionalitäten für die persistente Datenspeicherung und stellt grundlegende, für alle Requests notwendige Methoden zur Verfügung. Diese sind, neben der Verifikation der Requestsignatur, für die Aktualisierung der Attribute verantwortlich. Seine Hauptaufgabe besteht jedoch in der Delegierung der Arbeitsmethoden zu den folgenden spezialisierten Requests.

X509CrtRequest

Die Klasse `X509CrtRequest` beschreibt einen X509v3 Certification-Request und erzeugt ein Zertifikat.

X509CrlRequest

Die Klasse `X509CrlRequest` beschreibt einen X509v3 Certificate-Revocation-Request.

AdminRequest

Die Klasse `AdminRequest` beschreibt einen in die CA einzubringenden neuen Administrator .

IssuerRequest

Die Klasse `IssuerRequest` beschreibt einen in die CA einzubringenden neuen Issuer.

Types

Die Klasse `Types` beinhaltet eine Auflistung aller in der CA implementierten Requesttypen. Die in ihr definierten Konstanten werden von den Requests und Workbenches zur Identifizierung verwendet.

3.5.5 Geschäftsklassen der Domain Stock

Im Paket `Stock` sind die zentralen Klassen des Stock mit den notwendigen Funktionalitäten zur Persistenzverwaltung zusammengefaßt.

Die in Abbildung 3.10 dargestellten Klassen werden in den nachfolgenden Abschnitten beschrieben.

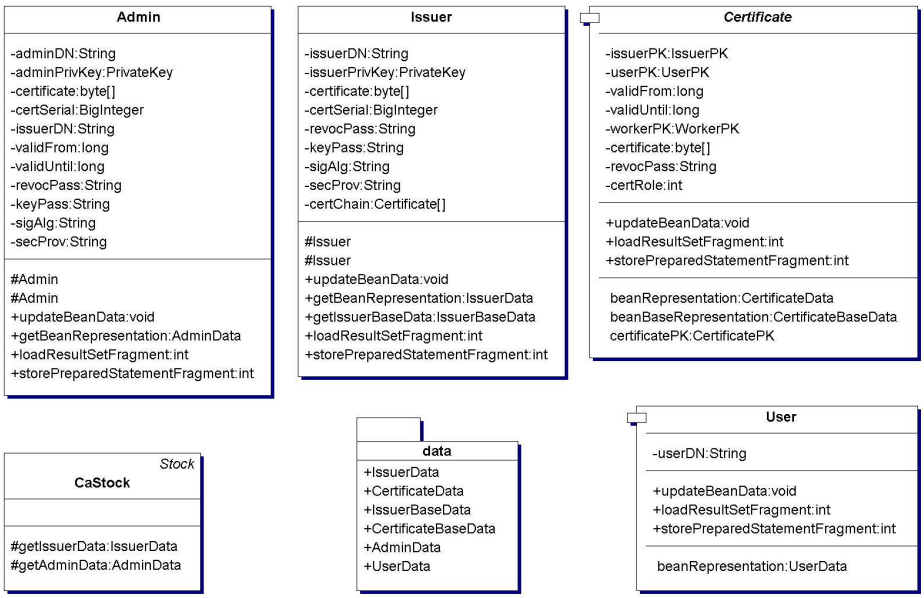

Abbildung 3.10: Geschäftsklassen der Domain Stock

Klasse Admin

Diese Klasse beschreibt einen Administrator. Neben den beschreibenden Attributen ist sie dafür zuständig, die Daten des Administrators zum Signieren ausgehender und zum Verifizieren eingehender Requests bereitzustellen.

Klasse CaStock

Diese Klasse stellt die zentrale Verwaltung persistenter Daten der CA dar.

Klasse Certificate

Die Klasse `Certificate` beschreibt ein X509v3 Zertifikat innerhalb der FlexiTRUST-CA.

Klasse Issuer

Diese Klasse beschreibt einen Issuer. Neben den beschreibenden Attributen ist sie dafür zuständig, Zertifikatsanträge zu unterschreiben.

Klasse User

Diese Klasse beschreibt einen User. Zu jedem Zertifikat muß auch der zugehörige User in der CA bekannt sein, um das Zertifikat zuordnen zu können.

Kapitel 4

Systemarchitektur

MW

Die Systemarchitektur der FlexiTRUST-CA basiert auf der Java 2 Platform Enterprise Edition (J2EE) [Inc02a]. J2EE ist eine von Sun Microsystems definierte und standardisierte Plattform für die Entwicklung von Java basierten mehrschichtigen (multi-tier) Geschäftsanwendungen. Sie definiert ein klares Architekturmodell und bietet eine einheitliche Plattform für die Erstellung, die Integration und den Betrieb der Anwendungen.

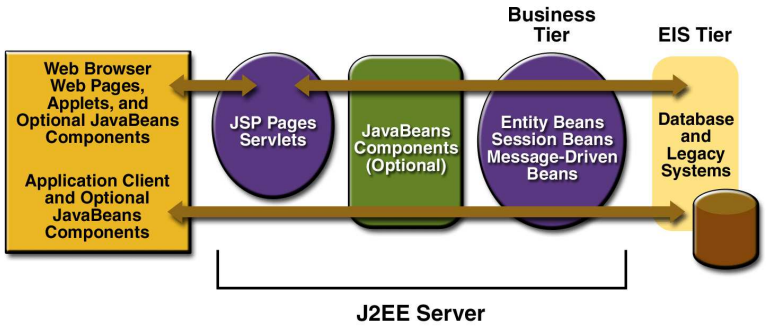
Business Tier
EIS Tier
Web Browser Web Pages, Applets, and Optional JavaBeans Components
Application Client and Optional JavaBeans Components
JSP Pages Servlets
JavaBeans Components (Optional)
Entity Beans Session Beans Message-Driven Beans
Database and Legacy Systems
J2EE Server

Abbildung 4.1: Business Tier und EIS Tier

J2EE unterscheidet das EIS-Tier[1] mit externen (legacy) Systemen, zu denen auch relationale Datenbanksysteme gerechnet werden, das Middle-Tier (hier als J2EE Server zusammengefasst), in dem z.B. die Business-Logik implementiert ist, und das Client-Tier. J2EE stellt insbesondere Komponenten für das Middle-Tier zur Verfügung. Im einzelnen sind das Enterprise-JavaBeans (EJB), Servlets, Java-Server-Pages (JSP) und unterstützende Dienste [Inc02b].

[1] Enterprise-Information-System

EJBs sind das Kernstück der J2EE. Sie ermöglichen die Erstellung von robusten, objektorientierten, verteilten Anwendungen. EJBs modellieren im wesentlichen Geschäftsobjekte oder (über Verwaltungsobjekte) Funktionalitäten. Sie werden in einem EJB-Container verwaltet, der den Enterprise-JavaBeans definierte Schnittstellen zu den vom Applikationsserver verwalteten Ressourcen anbietet. Zu diesen Diensten gehören z.B. Security, Transaktionsmanagement und Objektpersistenz. Darüber hinaus stehen die - nicht unbedingt an die Server-Seite gebundenen - Dienste der Java Plattform wie Java-Naming- and Directory-Interface (JNDI), Transaction-Service (JTA) [Inc02e] und Java Authentication and Authorization Service (JAAS) [Inc02d] etc. zur Verfügung. Die verbindliche Definition der Schnittstellen in der J2EE-Spezifikation stellt sicher, daß EJBs in verschiedenen Implementierungen von Applikationsservern unverändert genutzt werden können. Die Clients können - ebenfalls über wohldefinierte, vom Applikationsserver bereitzustellende Schnittstellen - auf die EJBs zugreifen. Für nähere Informationen zu Enterprise-JavaBeans und ihrer Modellierung wird auf die Studienarbeit [Win02] verwiesen.

Servlets sind ein stabiler und sicherer Mechanismus für die Ausführung von serverseitigem Code in Web-basierten Anwendungen. Servlets sind Klassen, die als "kleine Java Programme" die Funktionalität von Web-Servern erweitern und damit die Generierung von dynamischen Web-Seiten in Sprachen wie HTML, WML oder XML ermöglichen. Innerhalb der Servlets besteht die Zugriffsmöglichkeit auf spezifische Funktionalitäten des Web-Servers wie z.B. Session-Verwaltung aber auch auf sämtliche Java-APIs der J2EE und damit insbesondere auf EJBs.

Die JavaServer-Pages bilden ein "Front-End" zu Servlets. In JSPs kann statischer Text einer Web-Seite mit Java-Code für die Erzeugung der dynamischen Daten gemischt werden. Während der statische Anteil typischerweise das Layout beschreibt, können die Inhalte über entsprechenden Java-Code z.B. aus einer Datenbank geholt und in die generierte Seite integriert werden. JSPs trennen damit das Layout des User-Interface von den darunter liegenden dynamischen Daten. JSPs werden beim ersten Aufruf (und bei Änderung des JSP-Source-Codes) automatisch in Servlets übersetzt. Das bedeutet, daß innerhalb von JSPs die gleiche Umgebung wie in Servlets zur Verfügung steht und daß sich aus der Verwendung einer relativ abstrakten Sprache zur Seitenbeschreibung kein Performanceverlust gegenüber einer Implementierung in Java ergibt.

Aufbauend auf dieser Plattform ergibt sich für die FlexiTRUST-CA folgende, in Abbildung 4.2 dargestellte, grundsätzliche Struktur für das Gesamtsystem.

Die nachfolgenden Abschnitte beschreiben die Komponenten der FlexiTRUST-CA und ihre Aufgaben entsprechend ihrer Zuordnung zu den verschiedenen Schichten der J2EE-Architektur.

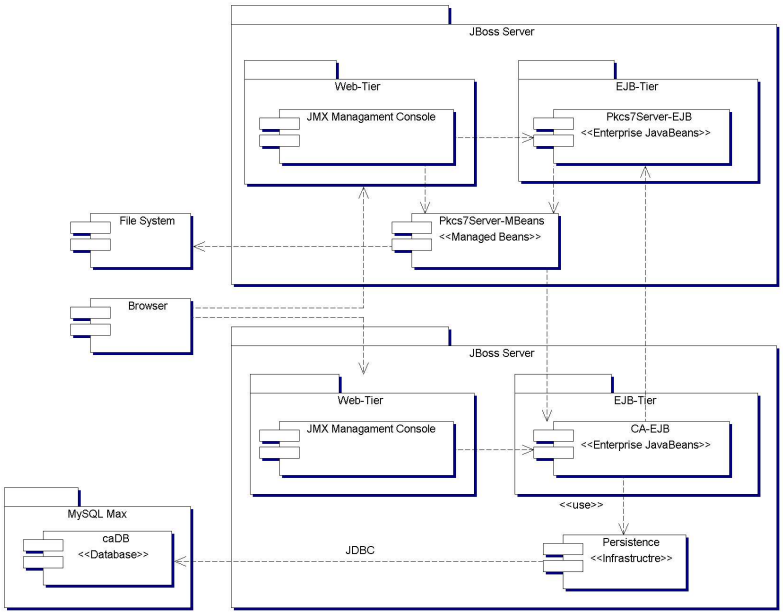

Abbildung 4.2: Struktur FlexiTRUST-CA

4.1 Architektur EIS-Tier

Im Projekt FlexiTRUST-CA beinhaltet das EIS-Tier ausschließlich eine Datenbank.

Die Datenbank enthält alle Daten, die im Rahmen der Requestbearbeitung der Anwendung FlexiTRUST-CA anfallen. Hierzu gehören beispielsweise Antragssteller, ausgestellte Zertifikate, Revokationen, Administratoren und Issuer. Der Zugriff auf die Datenbank erfolgt ausschließlich über Enterprise-JavaBeans. Die Beschreibung der Struktur der Datenbank ist in Kapitel 6 „Datenmodell“ enthalten.

4.2 Architektur EJB-Tier

Das EJB-Tier enthält in verschiedenen Paketen Enterprise-JavaBeans für die Themenbereiche `Bench`, `Entrance`, `Exit`, `Request` und `Stock`. Das Design der Entity-Beans und Session-Beans basiert auf der EJB-Spezifikation Version 2.0 .

4.2.1 Bench

Im Paket `Bench` sind die Enterprise-JavaBeans der Workbenches zusammengefasst. In Abbildung 4.3 ist die Session-Bean `CryptoBenchEJB` dargestellt.

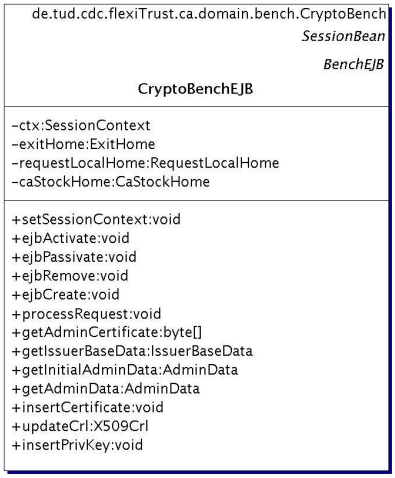

Abbildung 4.3: Enterprise JavaBeans Bench

Die in Abbildung 4.3 dargestellte Klasse wird in dem nachfolgenden Abschnitt beschrieben. Im Rahmen einer Erweiterung der CA sind die Enterprise-JavaBeans neuer Workbenches innerhalb dieses Packages zu implementieren (vgl. hierzu Kapitel 9 „Erweiterung des Systems“).

Klasse CryptoBenchEJB

Die Stateless-Session-Bean `CryptoBenchEJB` beinhaltet alle Operationen, die für das Verifizieren und Signieren in Verbindung mit weiteren Enterprise-JavaBeans notwendig sind. Sie stellt daher ein Bindeglied zwischen der in der Domänenklasse implementierten Logik und dem clusterweiten EJB-Tier der FlexiTRUST-CA dar. Sie ist eine Spezialisierung der Klasse `CryptoBench` aus der Domäne.

Über ihre Methode `processRequest` akzeptiert sie den Primary-Key eines Requests und instantiiert eine Repräsentation für die Bearbeitung innerhalb der eigenen Java-Virtual-Machine. Nach Abschluss der Arbeitsschritte des Requests bezüglich der Funktionalitäten der `CryptoBenchEJB` leitet diese den Primary-Key des Requests an die nächste von ihm definierte Bench weiter.

4.2.2 Entrance

Im Paket `Entrance` sind die Enterprise-JavaBeans des Entrance zusammengefaßt. In Abbildung 4.4 ist die Session-Bean `EntranceEJB` dargestellt.

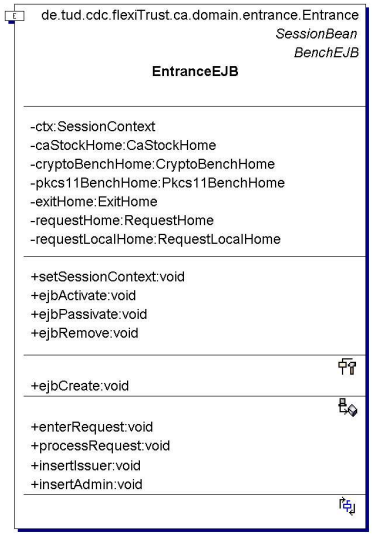

Abbildung 4.4: Enterprise JavaBeans Entrance

Die in Abbildung 4.4 dargestellte Klasse wird in dem nachfolgenden Abschnitt beschrieben.

Klasse EntranceEJB

Die Stateless-Session-Bean `EntranceEJB` beinhaltet momentan alle Operationen, die für die Bearbeitung von Issuer- und Admin-Requests in Verbindung mit weiteren Enterprise-JavaBeans notwendig sind (siehe Abschnitt 10.1.3 „CaStock“). Sie stellt daher ein Bindeglied zwischen der in der Domänenklasse implementierten Logik und dem clusterweiten EJB-Tier der FlexiTRUST-CA dar. Sie ist eine Spezialisierung der Klasse `Entrance` aus der Domäne.

Über ihre Methode `processRequest` akzeptiert sie den Primary-Key eines Requests. Im Falle der oben genannten Typen instantiiert die `EntranceEJB` eine Repräsentation für die Bearbeitung innerhalb der eigenen Java-Virtual-Machine. Nach Abschluss der Arbeitsschritte des Requests bezüglich der Funktionalitäten der `EntranceEJB` leitet diese den Primary-Key des Requests an die `ExitEJB` weiter. Alle anderen Request-Typen werden direkt an von ihnen spezifizierte Benches geschickt.

4.2.3 Exit

Im Paket `Entrance` sind die Enterprise JavaBeans des Exit zusammengefasst. In Abbildung 4.5 ist die Session-Bean `ExitEJB` dargestellt.

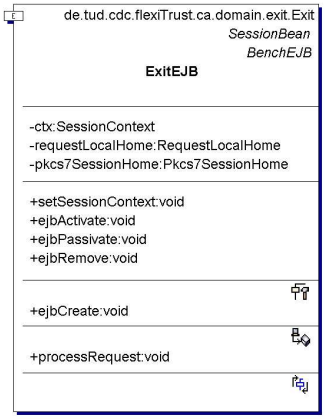

Abbildung 4.5: Enterprise JavaBeans Exit

Die in Abbildung 4.5 dargestellte Klasse wird in dem nachfolgenden Abschnitt beschrieben.

Klasse ExitEJB

Die Stateless-Session-Bean `ExitEJB` beinhaltet alle Operationen, die für abschließende Arbeitsschritte der Requests notwendig sind. Sie ist eine Spezialisierung der Klasse `Exit` aus der Domäne.

Über ihre Methode `processRequest` akzeptiert sie den Primary-Key eines Requests und ruft dessen Business-Methode `leave` auf. Der Request hat hier die Möglichkeit, sich auf das Verlassen der FlexiTRUST-CA vorzubereiten. Anschließend übergibt die `ExitEJB` den Primary-Key des abgeschlossenen Requests an den `Pkcs7Server` (vgl. Abschnitt 4.4.2 „Architektur Pkcs7Server“).

4.2.4 Request

Im Paket `Request` sind die Enterprise-JavaBean des Requests und die dazugehörigen Hilfsklassen zusammengefaßt. In Abbildung 4.6 ist die Entity-Bean `RequestEJB`, ihr

Data-Access-Object (DAO) `RequestDAO` und die Primary-Key-Klasse `RequestPK` dargestellt. Für nähere Informationen zu dem Designpattern Data-Access-Object sei auf [Inc02g] und [Win02] verwiesen.

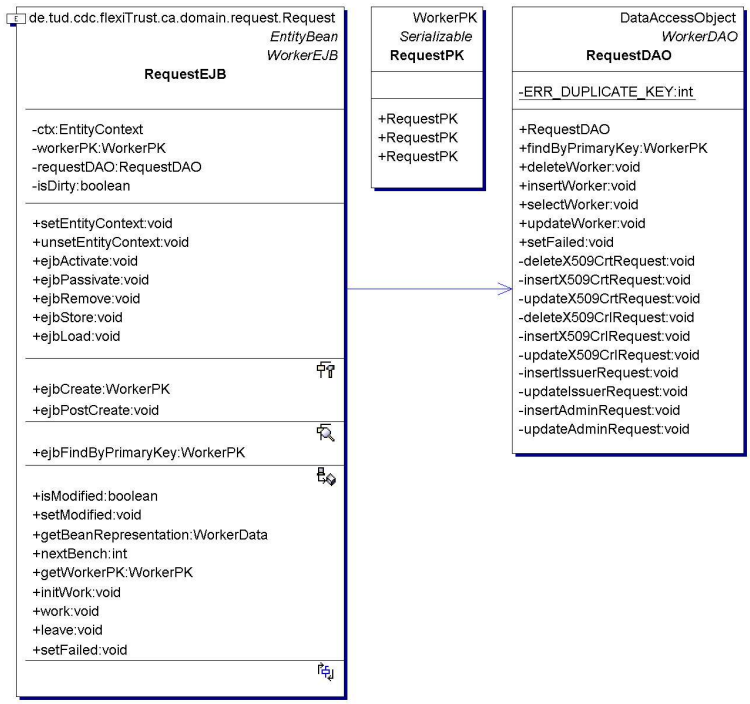

Abbildung 4.6: Enterprise JavaBeans Request

Die in Abbildung 4.6 dargestellten Klassen werden in den nachfolgenden Abschnitten beschrieben.

Klasse RequestEJB

Die Entity-Bean `RequestEJB` enthält die Daten eines einzelnen Requests im EJB-Tier. Sie ist eine Spezialisierung der Klasse `Request` aus der Domäne. Die Enterprise-JavaBean kann Darstellungen des Requests in Form von JavaBean-Komponenten liefern, sowie Daten auch umgekehrt wieder in die EJB einbringen. Bei diesen JavaBean-Komponenten handelt es sich um Klassen aus dem Paket `de.tud.cdc.flexiTrust.ca.domain.request.data` (vgl. hierzu auch Abschnitt 4.5.7 „Value Objects“).

Klasse RequestDAO

Die Klasse RequestDAO ist die Implementierung des Data-Access-Objects des Requests und zeichnet sich für alle mit der persistenten Datenspeicherung zusammenhängenden Pflichten verantwortlich. Abhängig von der Art der Requests stellt sie SQL-Statements zur Verfügung, deren Befüllung an die Domänen-Klasse `Request` und die spezialisierten Requests delegiert wird.

Klasse RequestPK

Die Klasse `RequestPK` beschreibt den internen Primary-Key eines Requests.

Der Lebenszyklus eines Requests

Im folgenden wird dargestellt, welche Schritte durchlaufen werden, wenn ein Request auf einer Bench zur Bearbeitung eintrifft:

1. Die `processRequest`-Methode der Bench wird mit dem Primary-Key des Requests aufgerufen.
2. Die Bench weist den Container über `findByPrimaryKey` auf das LocalHome-Interface der `RequestEJB` an, eine lokale Instanz der Request-Entity zu erzeugen.
3. Der Container ruft `ejbLoad` der `RequestEJB` auf. Diese greift auf die Datenbank zu und führt ein `select` auf alle dem spezifischen Request zugehörigen Attribute aus. Die Auswertung des ResultSets wird an die Geschäftsklasse der `RequestEJB`, der `domain.request.Request` übergeben. Hierzu wird die Methode `loadResultSetFragment` aufgerufen. Der allgemeine Teil des ResultSets wird von der Geschäftsklasse direkt ausgewertet, die spezifischen Attribute von der Spezialisierung. Anschließend liefert der Container der Bench eine Referenz auf das LocalInterface der Request-Entity zurück.
4. Die Bench ruft die Methode `work` des Requests auf und übergibt ihm hierbei als Parameter eine Referenz auf sich selbst.
5. Die Superklasse `Request` der `RequestEJB` leitet den Aufruf an den Request weiter, welcher den nächsten anstehenden Arbeitsschritt ausführt. Während dieser Zeit hat er vollen Zugriff auf die Public-Methoden der Geschäftsklasse der Bench.

6. Nachdem der Request den Arbeitsschritt durchgeführt hat, sind seine Attribute nicht mehr mit den Daten in der Datenbank identisch. Daher ruft der Container die Methode `ejbStore` der `RequestEJB` auf, die analog zu `ejbLoad` die aktualisierten Daten persistent in die Datenbank einbringt.

7. Anschließend fragt die Bench den Request an, welches die nächste Bench ist und ruft deren `processRequest`-Methode auf.

4.2.5 Stock

Im Paket `Stock` sind die Enterprise-JavaBeans des Stocks und die dazugehörigen Hilfsklassen zusammengefaßt. In den Abbildungen 4.7 und 4.8 sind die Entity-Beans `AdminEJB`, `CertificateEJB`, `IssuerEJB` und `UserEJB`, ihre Data-Access-Objects `AdminDAO`, `CertificateDAO`, `IssuerDAO` und `UserDAO` und die Primary-Key-Klassen `AdminPK`, `CertificatePK`, `IssuerPK` und `UserPK` dargestellt. Weiterhin ist die Session-Bean `CaStockEJB` mit ihrem DataAccessObject `CaStockDAO` aufgeführt.

Die in den Abbildungen 4.7 und 4.8 dargestellten Klassen werden in den nachfolgenden Abschnitten beschrieben.

Klasse AdminEJB

Die Entity-Bean `AdminEJB` enthält die Daten eines einzelnen Administrators im EJB-Tier. Sie ist eine Spezialisierung der Klasse `Admin` aus der Domäne. Die EJB kann eine Darstellung des Administrators in Form einer JavaBean-Komponente liefern, sowie Daten auch umgekehrt wieder in die EJB einbringen. Bei dieser JavaBean-Komponente handelt es sich um die Klasse `AdminData` aus dem Paket `de.tud.cdc.flexiTrust.ca.domain.stock.data`.

Klasse AdminDAO

Die Klasse `AdminDAO` ist die Implementierung des Data-Access-Objects des Administrators und zeichnet sich für alle mit der persistenten Datenspeicherung zusammenhängenden Pflichten verantwortlich. Sie stellt SQL-Statements zur Verfügung, deren Befüllung an die Domänen-Klasse `Admin` delegiert wird.

Klasse AdminPK

Die Klasse `AdminPK` beschreibt den internen Primary-Key eines Administrators.

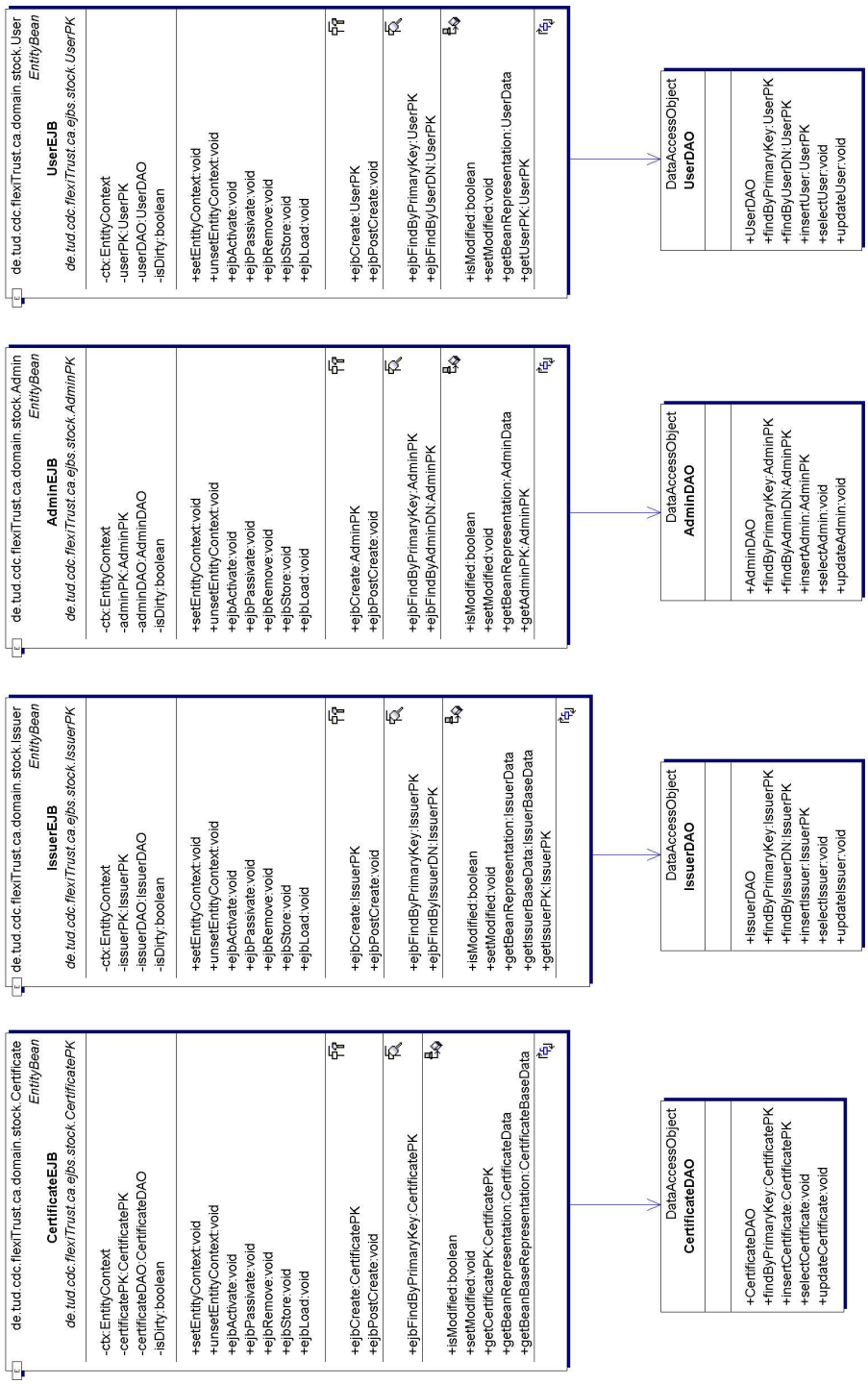

Abbildung 4.7: Enterprise JavaBeans Stock

Klasse CertificateEJB

Die Entity-Bean `CertificateEJB` enthält die Daten, die mit einem einzelnen X509v3 Zertifikats im EJB-Tier zusammenhängen. Sie ist eine Spezialisierung der Klasse `Certificate` aus der Domäne. Die EJB kann zwei Darstellungen des Zertifikats in Form von JavaBean-Komponenten liefern, sowie Daten auch umgekehrt wieder in die EJB einbringen. Bei diesen JavaBean-Komponenten handelt es sich um die Klassen `CertificateData` und `CertificateBaseData` aus dem Paket `de.tud.cdc.flexiTrust.ca.domain.stock.data`, wobei die Klasse `CertificateBaseData` nur Basisdaten enthält.

Klasse CertificateDAO

Die Klasse `CertificateDAO` ist die Implementierung des Data-Access-Objects des Zertifikats und zeichnet sich für alle mit der persistenten Datenspeicherung zusammenhängenden Pflichten verantwortlich. Sie stellt SQL-Statements zur Verfügung, deren Befüllung an die Domänen-Klasse `Certificate` delegiert wird.

Klasse CertificatePK

Die Klasse `CertificatePK` beschreibt den internen Primary-Key eines Zertifikats der gleichzeitig die Seriennummer des X509v3 Zertifikats ist.

Klasse IssuerEJB

Die Entity-Bean `IssuerEJB` enthält die Daten eines einzelnen Issuers im EJB-Tier. Sie ist eine Spezialisierung der Klasse `Issuer` aus der Domäne. Die EJB kann zwei Darstellungen des Issuers in Form von JavaBean-Komponenten liefern, sowie Daten auch umgekehrt wieder in die EJB einbringen. Bei diesen JavaBean-Komponenten handelt es sich um die Klassen `IssuerData` und `IssuerBaseData` aus dem Paket `de.tud.cdc.flexiTrust.ca.domain.stock.data`, wobei die Klasse `IssuerBaseData` nur Basisdaten enthält.

Klasse IssuerDAO

Die Klasse `IssuerDAO` ist die Implementierung des Data-Access-Objects des Issuers und zeichnet sich für alle mit der persistenten Datenspeicherung zusammenhängenden Pflichten verantwortlich. Sie stellt SQL-Statements zur Verfügung, deren Befüllung an die Domänen-Klasse `Issuer` delegiert wird.

Klasse IssuerPK

Die Klasse `IssuerPK` beschreibt den internen Primary-Key eines Issuers.

Klasse UserEJB

Die Entity-Bean `UserEJB` enthält die Daten eines einzelnen Users im EJB-Tier. Sie ist eine Spezialisierung der Klasse `User` aus der Domäne. Die EJB kann eine Darstellung des Users in Form einer JavaBean-Komponente liefern, sowie Daten auch umgekehrt wieder in die EJB einbringen. Bei dieser JavaBean-Komponente handelt es sich um die Klasse `UserData` aus dem Paket `de.tud.cdc.flexiTrust.ca.domain.stock.data`.

Klasse UserDAO

Die Klasse `UserDAO` ist die Implementierung des Data-Access-Objects des Users und zeichnet sich für alle mit der persistenten Datenspeicherung zusammenhängenden Aufgaben verantwortlich. Sie stellt SQL-Statements zur Verfügung, deren Befüllung an die Domänen-Klasse `User` delegiert wird.

Klasse UserPK

Die Klasse `UserPK` beschreibt den internen Primary-Key eines Users.

Klasse CaStockEJB

Die Stateless-Session-Bean `CaStockEJB` beinhaltet alle Operationen, die für das Speichern und Auslesen von Daten notwendig sind. Sie ist eine Spezialisierung der Klasse `CaStock` aus der Domäne.

Sie ist die zentrale Instanz für die Verwaltung von Issuern, Administratoren, Usern, Zertifikaten und Revokationen.

Klasse CaStockDAO

Die Klasse `CaStockDAO` ist die Implementierung des Data-Access-Objects des CaStock und zeichnet sich für alle mit der persistenten Datenspeicherung zusammenhängenden Pflichten verantwortlich.

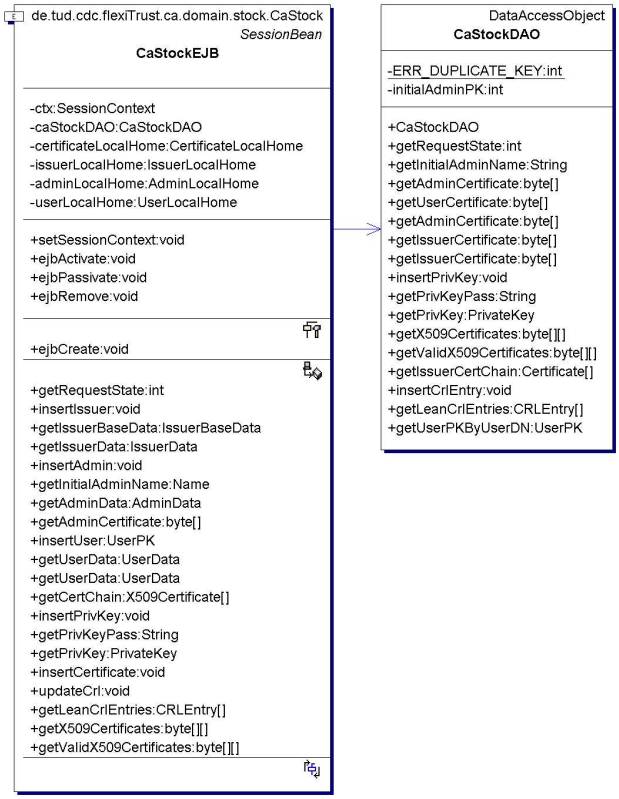

Abbildung 4.8: Enterprise JavaBeans CaStock

4.3 Architektur Web-Tier

Das Web-Tier beinhaltet im derzeitigen Entwicklungsstand dieser Certification-Authority ein JMX- und Unittest-Frontend. Der Zugriff kann mit einem beliebigen Browser über folgende Adressen erfolgen:

für das JMX-Frontend

```
http://localhost:8080/jmx.browser
http://localhost:8080/jmx-console
```

und für die Unittests mit

```
http://localhost:8080/requesttests
http://localhost:8080/stocktests
```

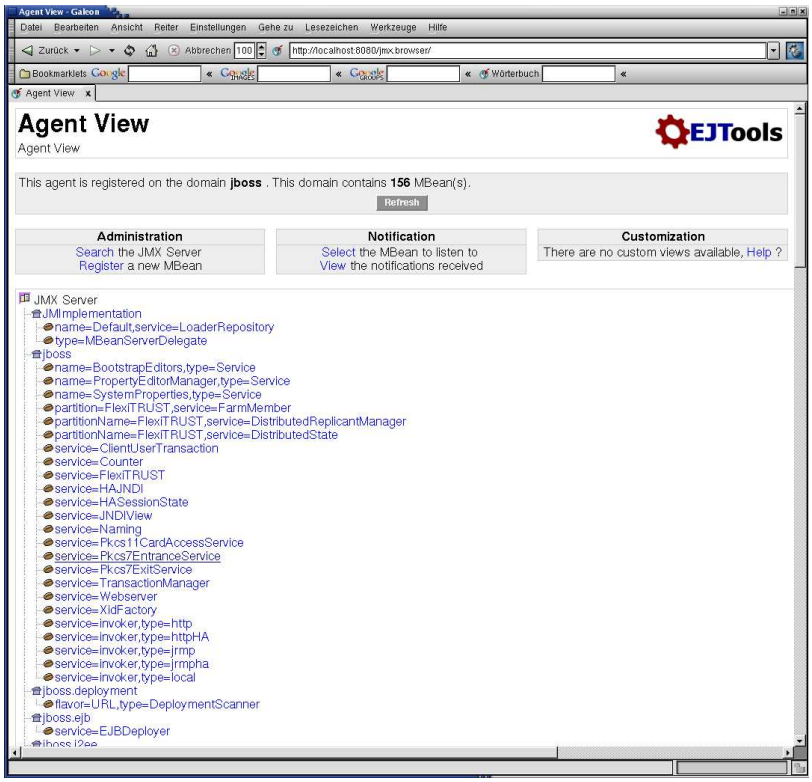

Abbildung 4.9: JMX.Browser Agent View

4.4 Architektur Client-Tier

Das Client-Tier basiert auf Dateiaustausch über definierte Verzeichnisse und auf der Verwendung von Web-Browsern. Weitere Clients sind nicht vorgesehen.

Für den Dateiaustausch werden vom Administrator auf einem oder mehreren Nodes Verzeichnisse für den Ein- und Ausgang von Requests vorgesehen. Dieses wird für PKCS#7-Requests durch den im folgenden beschriebenen Pkcs7Server durchgeführt. Die dazugehörigen Anwendungsfälle werden im nächsten Abschnitt beschrieben.

4.4.1 Anwendungsfälle des Pkcs7Server

Der `Pkcs7Server` untergliedert sich in die drei Komponenten `Entrance`, `Exit` und `Session`, deren Anwendungsfälle in den folgenden Abschnitten beschrieben werden.

Anwendungsfälle der Domain Entrance

Die nachfolgende Abbildung 4.10 zeigt alle Anwendungsfälle der Komponente `Entrance` des `Pkcs7Servers` und ihre Beziehungen untereinander.

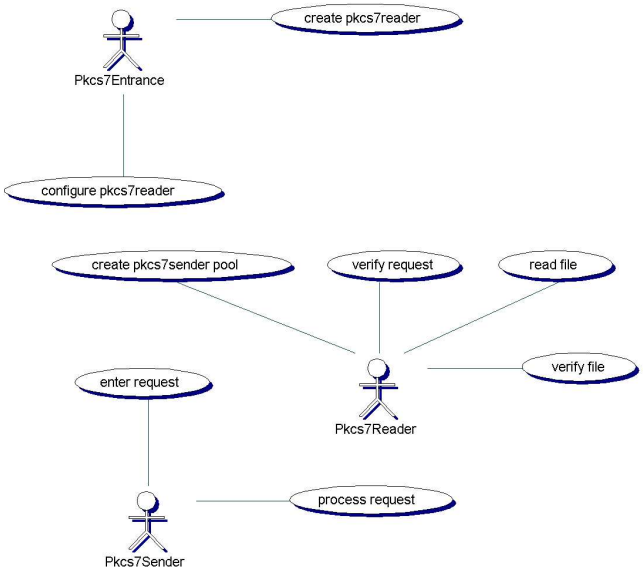

Abbildung 4.10: Anwendungsfälle des Pkcs7Server Entrance

Diese beschreiben die vom System durchzuführenden Anwendungsfälle für das Einlesen und die Verifikation der signierten Request-Packages. Weiterhin initiiert der Pkcs7-Server-Entrance die Bearbeitung der Requests im EJB-Tier der FlexiTRUST-CA.

Es können die drei Hauptkomponenten `Pkcs7Entrance`, `Pkcs7Reader` und `Pkcs7-Sender` unterschieden werden. Der `Pkcs7Entrance` ist für die Konfiguration dieses Teils des Pkcs7Servers verantwortlich, der `Pkcs7Reader` bearbeitet die Packages und der `Pkcs7Sender` initiiert die Bearbeitung.

Anwendungsfälle Bereich Pkcs7Entrance

- `create pkcs7reader`: Der `Pkcs7Entrance` ist für den Lebenszyklus des Readers verantwortlich. Seine primären Aufgaben sind die Requests einzulesen, sie zu verifizieren und in die CA einzubringen.
- `configure pkcs7reader`: In diesem Fall wird der Reader mit vom Administrator festgelegten Properties konfiguriert.

Anwendungsfälle Bereich Pkcs7Reader

- `create pkcs7sender pool`: Der `Pkcs7Reader` besitzt einen Thread-Pool variabler Größe mit Sendern für die weitere Verarbeitung. Zwischen den in Bearbeitung befindlichen Requests und den Threads besteht eine 1-zu-1-Beziehung.
- `read file`: Der `Pkcs7Reader` liest eine signierte Datei mit zu bearbeitenden Requests von der Festplatte ein.
- `verify file`: Der `Pkcs7Reader` verifiziert die Signatur der eingelesenen Datei.
- `verify request`: Der `Pkcs7Reader` verifiziert die in der eingelesenen Datei verpackten Requests und übergibt sie nacheinander dem Sender.

Anwendungsfälle Bereich Pkcs7Sender

- `enter request`: Der `Pkcs7Sender` erzeugt eine Bean-Repräsentation des Requests innerhalb des EJB-Tiers. Der Request wird persistent in die Datenbank eingebracht.
- `process request`: Der `Pkcs7Sender` initiiert die Bearbeitung des Requests.

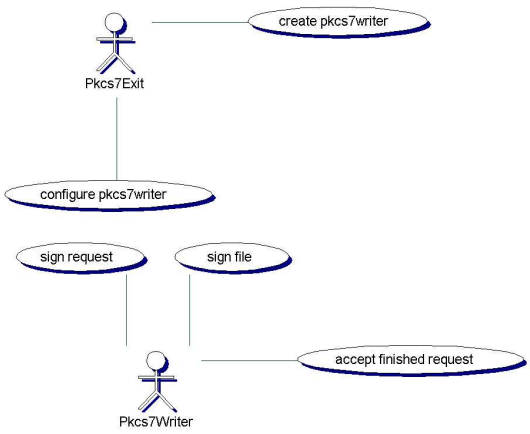

Abbildung 4.11: Anwendungsfälle des Pkcs7Server Exit

Anwendungsfälle der Domain Exit

Die nachfolgende Abbildung 4.11 zeigt alle Anwendungsfälle der Komponente `Exit` des `Pkcs7Servers` und ihre Beziehungen untereinander.

Diese beschreiben die vom System durchzuführenden Anwendungsfälle für das Signieren und Schreiben der bearbeiteten Requests, die das EJB-Tier der FlexiTRUST-CA durchlaufen haben.

Es können die Hauptkomponenten `Pkcs7Exit` und `Pkcs7Writer` unterschieden werden. Der `Pkcs7Exit` ist für die Konfiguration dieses Teils des Pkcs7Severs verantwortlich, der `Pkcs7Writer` signiert die abgearbeiteten Requests und erzeugt signierte Packages für die weitere Bearbeitung durch das Trustcenter.

Anwendungsfälle Bereich Pkcs7Exit

- `create pkcs7writer`: Der `Pkcs7Exit` ist für den Lebenszyklus des Writers verantwortlich. Seine primäre Aufgabe ist, die abgearbeiteten Requests zu signieren und in signierten Paketen in das Filesystem zu schreiben.
- `configure Pkcs7Writer`: In diesem Fall wird der Writer mit vom Administrator festgelegten Properties konfiguriert.

Anwendungsfälle Bereich Pkcs7Writer

- `accept finished request`: Nach Abschluß der Bearbeitung der Requests innerhalb des EJB-Tiers werden sie dem Writer von der Session-Bean `ExitEJB` übergeben.
- `sign request`: Der `Pkcs7Writer` signiert die abgearbeiteten Requests.
- `sign file`: Der `Pkcs7Writer` signiert vor dem Schreiben auf die Festplatte das Package mit einem oder mehreren Requests.

Anwendungsfälle der Domain Session

Die nachfolgende Abbildung 4.12 zeigt alle Anwendungsfälle der Komponente `Session` des `Pkcs7Servers` und ihre Beziehungen untereinander.

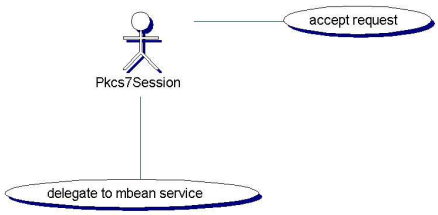

Abbildung 4.12: Anwendungsfälle des Pkcs7Server Session

Sie beschreiben vom System durchzuführende Anwendungsfälle bezüglich der Weiterleitung der Requests von dem EJB-Tier zu der Komponente `Exit`.

- `accept request`: Die `Pkcs7Session` akzeptiert den Primary-Key (PK) eines Requests, der innerhalb der CA abgearbeitet worden ist.
- `delegate to mbean service`: PKs werden an die Komponente `Exit` weitergeleitet.

4.4.2 Architektur Pkcs7Server

Der `Pkcs7Server` implementiert den Dateizugriff und stellt diesen der FlexiTRUST-CA als Service zur Verfügung. Derartige Services werden in J2EE als Managed-Beans (MBeans) realisiert [SJG02a]. Er untergliedert sich in die Komponenten `Entrance`, `Exit` und `Session`, welche im folgenden beschrieben werden.

Entrance

Im Paket `Entrance` sind die MBean-Service-Klassen des `Pkcs7Entrance` und die dazugehörigen Hilfsklassen zusammengefasst. In Abbildung 4.13 sind die MBean `Pkcs7-Entrance`, der `Pkcs7Reader` und die Klassen des `ThreadPools` dargestellt und in den nachfolgenden Abschnitten beschrieben.

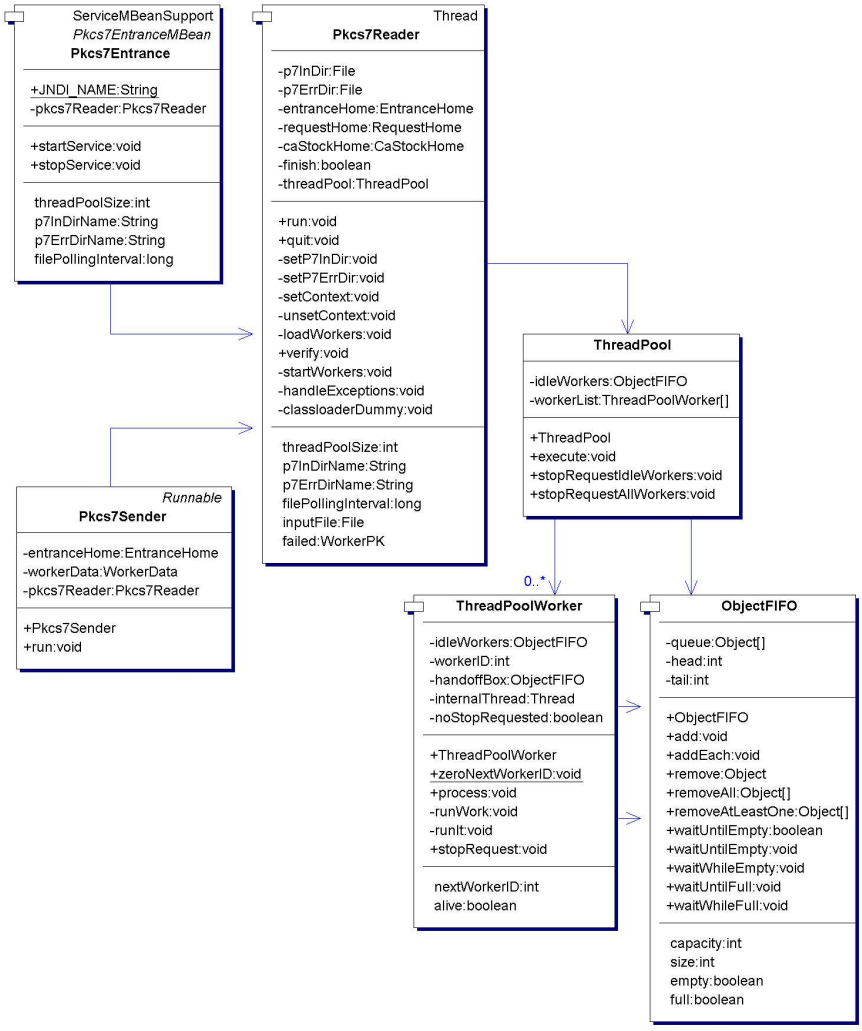

Abbildung 4.13: MBean Service Pkcs7Entrance

Klasse Pkcs7Entrance Die Klasse `Pkcs7Entrance` implementiert eine Managed-Bean. Sie hält eine Instanz des `Pkcs7Readers`, die von ihr zur Laufzeit konfiguriert

wird.

Klasse Pkcs7Reader Die Klasse `Pkcs7Reader` zeichnet sich für das Einlesen der von der FlexiTRUST-CA zu bearbeitenden Request-Packages im `p7`-Format verantwortlich. Diese Packages enthalten DER-encoded SignedData-Objekte, wobei jedes davon ebenfalls signiert ist und die Daten genau eines Request-Objekts enthält.

Die Request-Authority, ein Bestandteil der FlexiPKI-Infrastruktur, erzeugt Pakete mit Requests welche vom `Pkcs7Reader` über die Dateischnittstelle eingelesen und verifiziert werden. Der Reader packt für die weitere Verarbeitung die Requests aus und übergibt sie über den `Pkcs7Sender` einzeln dem EJB-Tier der CA für die Verarbeitung. Beim Start des `Pkcs7Readers` wird einmalig ein `ThreadPool` mit einer beliebigen aber festen Anzahl von `Pkcs7Sender`-Threads erzeugt. Die Benutzung eines Pools mit beschränkter (aber konfigurierbarer, vgl. Abschnitt 8.3 „Betrieb des Pkcs7Servers“) Anzahl von instantiierten Threads gewährleistet hierbei, daß mit steigender Zahl von Anfragen die Belastung des Systems begrenzt bleibt. Das Absehen von weiteren Instantiierungen und damit die sukzessive Wiederverwendung der Threads schont darüber hinaus die Ressourcen des Servers (vgl. hierzu Abschnitt 5.1 „Clustering“).

Der `Pkcs7Reader` überwacht in einstellbaren Intervallen das vom System-Administrator spezifizierte `p7In`-Verzeichnis (vgl. hierzu Abschnitte 7.3.3 „Installation der Anwendung“ und 8.3 „Betrieb des Pkcs7Servers“).

Klasse Pkcs7Sender Die Klasse `Pkcs7Sender` initiiert die Bearbeitung eines Requests durch das EJB-Tier. Es besteht eine 1-zu-1-Beziehung zwischen `Request` und `Pkcs7Sender`-Instanz. Innerhalb eines ThreadPools werden eine konfigurierbare Anzahl von Sender-Instanzen vorgehalten.

Klasse ThreadPool Die Klasse `ThreadPool` stellt dem `Pkcs7Reader` einen Pool von `Pkcs7Sendern` zur Verfügung. Durch die konfigurierbare Größe dieses Pools wird die maximale Anzahl der sich gleichzeitig innerhalb der FlexiTRUST-CA befindlichen Requests beschränkt. Im Zusammenhang mit der Klasse `ThreadPool` stehen die Klassen `ObjectFIFO` und `ThreadPoolWorker`.

Klasse ThreadPoolWorker Die Klasse `ThreadPoolWorker` repräsentiert einen Thread innerhalb des Pools.

Klasse ObjectFIFO Die Klasse `ObjectFIFO` hält eine Liste von Objekten, die nach dem First-In-First-Out-Prinzip verwaltet wird.

Exit

Im Paket `Exit` sind die MBean-Service-Klassen des `Pkcs7Exit` und die dazugehörigen Hilfsklassen zusammengefasst. In Abbildung 4.14 ist die MBean `Pkcs7Exit` und der `Pkcs7Writer` dargestellt, die in den nachfolgenden Abschnitten beschrieben werden.

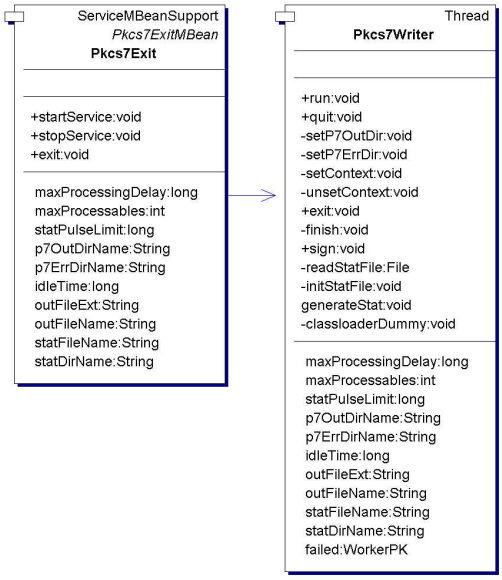

Abbildung 4.14: MBean Service Pkcs7Exit

Klasse Pkcs7Exit Die Klasse `Pkcs7Exit` implementiert eine Managed-Bean. Sie hält eine Instanz des `Pkcs7Writers`, die von ihr zur Laufzeit konfiguriert wird.

Klasse Pkcs7Writer Die Klasse `Pkcs7Writer` nimmt die von dem EJB-Tier bearbeiteten Requests entgegen und erzeugt mit dessen Daten ein SignedData-Objekt. Diese Objekte werden DER-encoded und wiederum in ein unterschriebenes SignedData-Objekt umgewandelt. Dieses Paket wird anschließend DER-encoded als Datei in das vom System-Administrator spezifizierte `p7Out`-Verzeichnis abgelegt (vgl. hierzu Abschnitte 7.3.3 „Installation der Anwendung“ und 8.3 „Betrieb des Pkcs7Servers“).

Session

Im Paket `Session` sind die Enterprise-JavaBean der `Pkcs7Session` und die dazugehörigen Hilfsklassen zusammengefaßt. In Abbildung 4.15 ist die Enterprise-JavaBean

`Pkcs7SessionEJB` dargestellt, die in dem nachfolgenden Abschnitt beschrieben.wird.

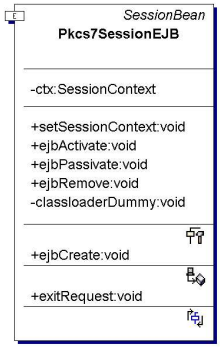

Abbildung 4.15: Enterprise JavaBeans Pkcs7Session

Klasse Pkcs7SessionEJB Die Klasse `Pkcs7SessionEJB` stellt die Schnittstelle zwischen dem EJB-Tier der FlexiTRUST-CA und dem `Pkcs7Server` dar.

4.5 Grundlagen und Begründungen für die vorgestellte Architektur

Während der Entwurfs- und Implementierungsphase der im Rahmen dieser Diplomarbeit entstandenen FlexiTRUST-CA wurden durch die Problemstellungen bedingte Entscheidungen getroffen. Einige der wesentlichen Aspekte werden hier im folgenden erläutert und begründet.

4.5.1 Local-Interfaces der Entity-Beans

Die FlexiTRUST-CA wurde als verteilte Anwendung entwickelt. Diesem Ansatz widerspricht jedoch die Verwendung von Local-Interfaces für Entity-Beans. Dies liegt darin begründet, daß bei Nutzung dieser Art Interfaces dem Container die Möglichkeit des Load-Balancings und Fail-Over genommen wird. Dem gegenüber steht der Performancegewinn durch lokale Aufrufe innerhalb der selben JVM, denn Remote-Calls sind aufgrund der Stubs, Skeletons, Netzwerkzugriffe, dem Marshalling/Demarshalling und dem EJB-Object-Interceptor sehr teuer [ER02a]. Daher sind in der hier vorliegenden Implementierung Workbenches als Session-Beans implementiert und fungieren als Wrapper für die Entity-Beans `RequestEJB`, `IssuerEJB`, etc. (vgl. Abbildung 4.16). Damit erfolgt Load-Balancing und Fail-Over über diese Session-Beans.

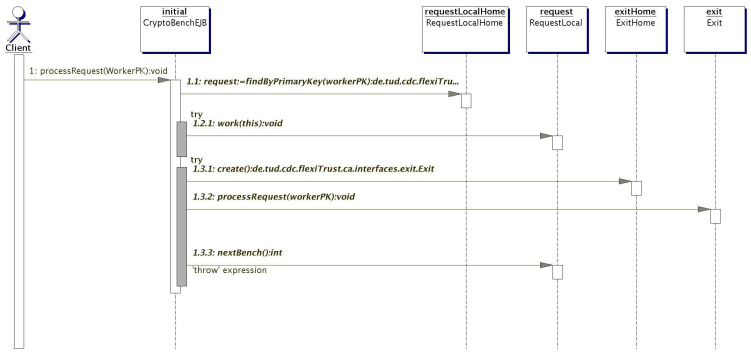

Abbildung 4.16: Sequenzdiagramm processRequest

4.5.2 HA-JNDI und die Pkcs7Server-Session

In dem Abschnitt 4.4.2 über die MBean-Services wurde die Session-Bean `Pkcs7SessionEJB` beschrieben, die einen sehr geringen Funktionsumfang aufweist. Warum diese jedoch notwendig ist, soll hier erläutert werden. Als Grundlage hierfür wird zunächst erklärt, wie das Java-Naming- und Directory-Interface (JNDI) in einem Cluster arbeitet.

Dem Zugriff auf eine Enterprise-JavaBean geht in den meisten Fällen ein Lookup auf deren Home-Interface innerhalb eines JNDI-Baumes voraus. Der Client hat im Anschluss daran die in dem Interface spezifizierten Möglichkeiten, an Instanzen dieser Bean zu gelangen. Im geclusterten Fall ist ein einfacher JNDI jedoch nicht ausreichend, da dieser keine Kenntnis von den Einträgen auf den anderen Nodes hat. Daher existiert hier ein clusterweit replizierter JNDI-Context, den die Clients für die Lookups verwenden und Objekte an ihn binden können. Dieser nodeübergreifende JNDI nennt sich im Fall des in diesem Projekt verwendeten Application-Servers High-Availability-JNDI (HA-JNDI) [SLJG02].

Der `Pkcs7Server` stellt Services für Festplattenzugriff zur Verfügung. Die dazugehörigen Objekte sind nicht serialisierbar. Daher können sie nicht an den HA-JNDI gebunden werden, wodurch sie für andere Nodes im Cluster nicht zugreifbar sind. Um dieses Problem zu umgehen, existiert die Stateless-Session-Bean `Pkcs7SessionEJB`, die als Wrapper für das EJB-Tier der FlexiTRUST-CA fungiert. Diese Session-Bean bindet sich an den globalen HA-JNDI an und stellt somit das Bindeglied zwischen dem lokalen JNDI, in den diese Objekte eingebunden werden, und HA-JNDI dar.

4.5.3 EJBServiceLocator

Enterprise-JavaBeans führen häufig Lookups auf von ihnen benötigte Service-Objekte, beispielsweise Home- oder Remote-Interfaces, über den (HA-)JNDI aus. Bei jedem Lookup wird ein InitialContext benötigt, der eventuell neu erzeugt werden muss. Insbesondere mit den sich daran anschließenen Object-Lookups geht ein signifikanter Performanceverlust einher. Um diesen zu minimieren, wurde hier der Ansatz des EJBServiceLocators gewählt, dessen Implementierung sich im Package `jws.util` befindet [Inc02h]. Diese Klasse stellt ein Singleton dar, d.h. sie existiert nur ein einziges mal pro JVM. Sie hält neben einem einmal erzeugten InitialContext interne Tabellen mit bereits angefragten EJB-Home-Interface-Objekten, die bei Bedarf wieder direkt zurückgeliefert werden können. Hierdurch werden unnötige Lookups effizient und einfach vermieden.

4.5.4 Transaction Handling

Transaktionen stellen ein einfaches Modell für Erfolg oder Mißerfolg eines Anwendungsfalles dar. Eine Transaktion kann nur auf zwei Arten beendet werden: Entweder werden alle Änderungen dauerhaft eingebracht (engl.: commit) oder alle Änderungen werden zurückgenommen (engl.: abort/rollback). Wird eine Transaktion „comittet“, müssen alle Aktionen innerhalb dieser Transaktion erfolgreich durchgeführt worden sein. Schlagen jedoch eine oder mehrere Aktionen fehl, wird für die gesamte Transaktion ein Rollback durchgeführt und alle Änderungen werden zurückgenommen. Das Modell einer Transaktion spezifiziert also ein Alles-oder-Nichts-Modell.

Folgende Eigenschaften werden definiert.

- Atomicity: Wird die Transaktion durch Fehler unterbrochen, werden alle bisherigen Auswirkungen rückgängig gemacht
- Consistency: Die Auswirkungen einer Transaktion bewahren die Konsitenz
- Isolation: Zwischenzustände sind für andere Transaktionen unsichtbar
- Durability: Die Auswirkungen einer ausgeführten Transaktion sind dauerhaft

Diese Eigenschaften zusammen werden als ACID-Prämissen bezeichnet.

Der von JBoss mitgelieferte Transaktionsmanager bietet Unterstützung für Dienste wie Abgrenzung einzelner Transaktionen und die Übermittlung des Transaktionskontexts. Die FlexiTRUST-CA wurde als verteilte Applikation entworfen und somit können Transaktionen mehrere Nodes umfassen. Da diese verteilten Transaktionen jedoch

vom JBoss-eigenen, nach JTA-Spezifikation implementierten, Transaktions-Monitor [SJG02b] nicht unterstützt werden, findet hier Tyrex 1.0 [Gro02a] als JBoss-Plugin Anwendung. Tyrex ist eine OpenSource-Implementierung der Java-Transaction-Service (JTS) Spezifikation [Inc02f], die von Sun Microsystems definiert wurde und verteilte Transaktionen berücksichtigt.

4.5.5 Unittests

> Dante could never have painted a more fearsome hell than the last few months of a large software project which did not make extensive use of unit testing.
>
> unknown

Für die Testfälle wird in diesem Projekt eine erweiterte Version von JUnit, die JUnitEE verwendet. Die Standardtestfälle mit JUnit arbeiten als Client des EJB-Tiers und laufen in einer getrennten JVM. Daher besitzen sie einen eigenen Context und es sind somit nur reine Black-Box-Tests möglich.

Unittests in Verbindung mit JUnitEE ermöglichen es nun, die Testfälle innerhalb des Application-Servers ablaufen zu lassen. Die Testfälle werden zusammen mit einem einfachen Web-Front-End in ein Web-ARchive (WAR-File) eingepackt, dynamisch durch den Server geladen und sind somit jederzeit über einen beliebigen Browser ausführbar. Der größte Vorteil jedoch ist, daß sie sich auch im gleichen Context befinden, wodurch die Interaktion der Beans untereinander testbar wird.

Für die hier implementierte FlexiTRUST-CA wurden Testfälle geschrieben, die das gesamte Leistungsspektrum der bislang umgesetzten Funktionen testen können.

Diese Tests untergliedern sich in die Hauptbereiche Request und CaStock, wobei die Workbenches ebenfalls mitgetestet werden. Der Zugriff auf die Unittests erfolgt mit einem Browser über die Adresse

```
http://localhost:8080/requesttests
```

bzw.

```
http://localhost:8080/stocktests.
```

Die folgende Tabelle gibt Aufschluß über die bislang implementierten Testfälle. Weitere können in dem Verzeichnis `$FLEXITRUST_HOME/ca/tst` und dessen Unterverzeichnissen implementiert werden.

Testname	Beschreibung
`TestAdminRequest`	Erzeugt einen neuen Administrator
`TestCertChain`	Fordert die Zertifikatskette eines Zertifikats an
`TestInitKeyRequest`	Erzeugt einen initialen Issuer und einen initialen Administrator
`TestInsertUser`	Erzeugt einen neuen User
`TestIssuerRequest`	Erzeugt einen neuen Issuer
`TestRequest`	Test der Basisfunktionalitäten der Request-Bean
`TestUserCertificates`	Fordert alle Zertifikate eines gegebenen Users an
`TestX509CrlRequest`	Erzeugt einen X509v3 Certificate-Revocation-Request
`TestX509CrtRequest`	Erzeugt einen X509v3 Certification-Request

4.5.6 XDoclet

XDoclet ist ein Java-Dienstprogramm, das die Verwaltung der Metadaten von Komponenten mit Hilfe von JavaDoc-ähnlichen Kommentaren in einer gemeinsamen Quelldatei ermöglicht und die Entwicklung von J2EE-Komponenten wie Enterprise-JavaBeans, MBeans u.a. auf diese Weise deutlich vereinfacht.

Am Beispiel einer Enterprise-JavaBean sei dies verdeutlicht: Zu einer EJB gehören neben der EJB-Implementierung Interfaces, welche die Zugriffsmöglichkeiten auf die Bean beschreiben. Dieses können Home-, Remote-, LocalHome- und Local-Interfaces sein. In einem Remote-Interface beispielsweise werden die Signaturen aller aufrufbaren Methoden der Bean aufgezählt. Bei bis zu vier Interfaces entsteht dadurch viel redundanter Code. Neben den Interfaces werden noch Deployment-Deskriptoren benötigt, die die Eigenschaften der Beans und ihre Anforderungen an die Laufzeitumgebung beschreiben. Dies umfasst u.a. die Spezifikation von Abhängigkeiten zwischen mitgelieferten und noch fehlenden Komponenten, sowie zu den benötigten Services und Ressourcen auf der noch unbekannten Zielplattform. XDoclet braucht normalerweise nur eine Datei, nämlich die EJB Implementierung, und generiert daraus alle weiteren notwendigen Dateien, um eine EJB zu deployen.

Nähere Informationen zu XDoclet können [AS02] entnommen werden.

Die Generierung der Interfaces und Deployment-Deskriptoren wird während des Buildvorgangs der FlexiTRUST-CA durch Ant-Targets durchgeführt.

4.5.7 Value Objects

Bei Enterprise-JavaBeans handelt es sich um Remote-Objekte, die einen signifikanten Anteil an Systemressourcen und Netzwerkbandbreite benötigen. Deshalb sollte ein

Geschäftsobjekt, das nur Daten hält und diese über Zugriffsmethoden dem Client zur Verfügung stellt, nicht als Enterprise Bean modelliert werden. Eine bessere Alternative sind sogenannte Datentransferobjekte (engl. Value-Object). Sie sind serialisierbar und können komplett als Werte zwischen Client und Server verschickt werden. Oft ist es sinnvoll, daß Entity-Beans einem Client ihre kompletten Daten durch nur einen Remoteaufruf in Form eines Value-Objects zur Verfügung stellen. Zusätzlich zu diesem Performancevorteil ergibt sich der Nebeneffekt, daß nicht sämtliche betroffenen Methodensignaturen geändert werden müssen, wenn das Value-Object erweitert wird.

Ein Beispiel hierfür ist das abschließende Verpacken eines Requests in eine signierte Datei durch den `Pkcs7Server`. Hierfür fragt der Pkcs7Server die Request-Entity nach ihrer Bean-Darstellung und ruft ihre Businessmethode `getBeanRepresentation` auf. Der `Pkcs7Server` erhält infolge dessen eine `WorkerData` mit allen die Bean spezifizierenden Attributen. Es ist nur ein Remote-Call notwendig, wodurch die Anzahl der Netzwerkzugriffe stark reduziert wird.

Die Implementierung der Value-Objects für die FlexiTRUST-CA befindet sich in den Packages `de.tud.cdc.flexiTrust.ca.domain.request.data` und `de.tud.cdc.flexiTrust.ca.domain.stock.data`.

Kapitel 5

Umsetzung der fachlichen Anforderungen

MW

In diesem Kapitel werden einzelne Aspekte der Implementierung behandelt, denen besondere Beachtung im Rahmen der Design- und Implementierungsphase geschenkt wurde.

5.1 Clustering

Das System ist als verteilte Anwendung ausgelegt, wodurch Ausfallsicherheit und Lastverteilung gewährleistet wird. Zur Laufzeit können weitere Nodes gestartet werden, welche sich automatisch in den bereits bestehenden Cluster einbinden. Die Verteilung der Aufgaben und somit der Last geschieht auf Ebene der als Session-Bean realisierten Elemente wie Benches und CaStock, wodurch das Clustering ausreichend genutzt und das System nicht durch das Suchen zahlreicher im Cluster verteilter Entities ausgebremst wird. Bei Ausfall eines oder mehrerer Nodes findet ein Failover statt. Hierbei werden gegebenenfalls sich in Bearbeitung befindliche Requests abgebrochen, die dann in der Datenbank mit Status `failed` markiert werden [ER02b].

Realisiert wird das Clustering durch einen clusterweit replizierten JNDI-Baum, eine kurze Beschreibung hiervon ist unter Abschnitt 4.5.2 „HA-JNDI und die Pkcs7ServerSession“ zu finden.

Da geclusterte Datenbanken noch in der Entwicklungsphase sind, greifen sämtliche Nodes auf einen Datenbankserver zu. Der Ansatz zu möglichst großer Ausfallsicherheit wird in Abschnitt 7.2.3 „Replikation der Datenbank“ beschrieben.

5.2 Logging

Zur Laufzeit der Anwendung sollen Informationen über den Programmablauf protokolliert werden. Hierfür kommt Log4J [Pro02] zum Einsatz, die dazugehörigen Statements werden mit Hilfe von AspectJ [Org02] generiert. Auch zur Laufzeit kann JBoss über die Log4J-Konfigurationsdatei `$JBOSS_HOME/server/flexiTRUST/conf/log4j.xml` mitgeteilt werden, welche Klassen von generierten Statements tatsächlich ausgewertet werden sollen.

Zum einen werden alle Fehler, die zur Laufzeit auftreten, in Log-Dateien geschrieben, zum anderen können beliebige Stellen im Quellcode mit Log-Statements versehen werden, um Variableninhalte auszugeben.

Im Einzelnen werden im folgende Größen ausgegeben:

- Zeitstempel (Zeitpunkt der Meldung),
- Meldungsklasse („Error“, „Warning“ oder „Info“),
- Meldung (detaillierte Beschreibung eines Fehlers oder Variableninhalte).

5.3 Schnittstellen zu externen Systemen

Aufgrund der verteilten Architektur der FlexiTRUST-CA müssen für den Dateizugriff besondere Vorkehrungen getroffen werden. Da nicht jeder Node Zugriff auf Dateisysteme besitzt, wird dieser über einen clusterweit verfügbaren Service realisiert. Umgesetzt wurde dieser Services durch den Pkcs7Server, der im Abschnitt 4.4.2 „Architektur Pkcs7Server“ beschrieben wird.

5.4 Konfi guration und Management

Die Konfiguration der Anwendung erfolgt durch eine Property-Datei. Diese wird beim Systemstart vom `SystemPropertiesService`, welcher Bestandteil des JBoss-Application-Servers ist, eingelesen und der CA zur Verfügung gestellt. Die Property-Datei und ihre einzelnen Konfigurationsparameter werden in Abschnitt 7.3.3 „Installation der Anwendung“ beschrieben.

Zur Laufzeit kann der Zustand der Anwendung über die JMX-Management-Console abgefragt werden.

Kapitel 6

Datenmodell

LF

In diesem Kapitel wird das verwendete Datenmodell beschrieben und erläutert.

6.1 Übersicht

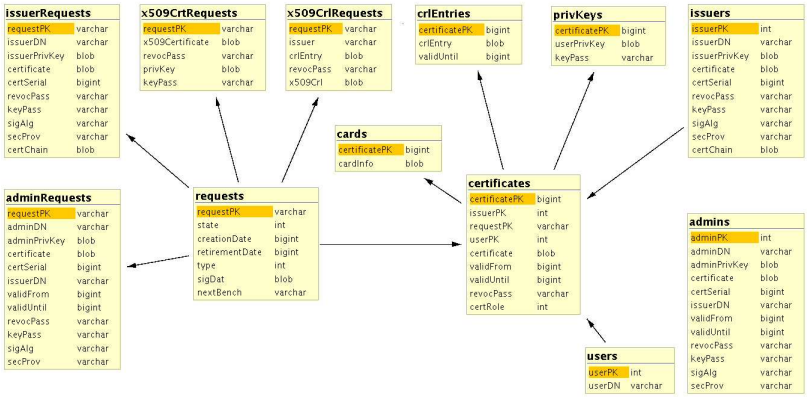

Abbildung 6.1: Datenmodell

6.2 Die Tabellen

Tabelle requests In dieser Tabelle werden alle requestspezifischen Daten gespeichert, die jeder Request benötigt. Dazu gehören der systemweit eindeutige Primärschlüssel des Requests, der Requeststatus, der Zeitpunkt der Erstellung, der Zeitpunkt,

zu dem eine zu lange andauernde Bearbeitung abgebrochen werden soll, der Typ des Requests, die signierten Daten des Requests und die nächste zu benutzende Bench.

Abhängig vom Requesttyp müssen weitere Daten gespeichert werden, dazu erhält jeder Requesttyp eine eigene Tabelle:

Tabelle issuerRequest Diese Tabelle enthält den Primärschlüssel des Requests als Fremdschlüssel zur korrekten Zuordnung zu den Daten in der Tabelle `requests`. Zusätzlich werden hier der Name, der Private Key und das Zertifikat des Issuers, das Paßwort zum Private Key, der Signaturalgorithmus des Issuers, das Paßwort zum Revozieren des Zertifikats, der verwendete Security Provider und die CertificateChain gespeichert.

Tabelle adminRequest Diese Tabelle enthält den Primärschlüssel des Requests als Fremdschlüssel zur korrekten Zuordnung zu den Daten in der Tabelle `requests`. Zusätzlich werden hier der Name, der Private Key und das Zertifikat des Administrators, das Paßwort zum Private Key, der Signaturalgorithmus des Administrators, das Paßwort zum Revozieren des Zertifikats und der verwendete Security Provider gespeichert.

Tabelle x509CrtRequests Diese Tabelle enthält den Primärschlüssel des Requests als Fremdschlüssel zur korrekten Zuordnung zu den Daten in der Tabelle `requests`. Zusätzlich wird hier das Zertifikat, das Paßwort zum Revozieren des Zertifikats, der Private Key des Users und das Paßwort zum Private Key gespeichert.

Tabelle x509CrlRequests Diese Tabelle enthält den Primärschlüssel des Requests als Fremdschlüssel zur korrekten Zuordnung zu den Daten in der Tabelle `requests`. Zusätzlich wird der Name des Issuers, der CRLEntry und das Revokationspaßwort des zu revozierenden Zertifikats gespeichert.

Bis zur Erweiterung der CA sind dies alle requestspezifischen Tabellen in der Datenbank. Desweiteren werden folgende Tabellen zur Verwaltung der CA-internen Daten benötigt:

Tabelle issuers In diese Tabelle werden nach erfolgreicher Bearbeitung eines IssuerRequests sämtliche zugehörigen Daten der Tabelle `issuerRequests` mit Ausnahme des Primärschlüssels des Requests übernommen, zusätzlich wird ein von der Datenbank automatisch hochgezählter Primärschlüssel für den Issuer vergeben und gespeichert.

Tabelle admins In diese Tabelle werden nach erfolgreicher Bearbeitung eines AdminRequests sämtliche zugehörigen Daten der Tabelle `adminRequests` mit Ausnahme des Primärschlüssels des Requests übernommen, zusätzlich wird ein von der Datenbank automatisch hochgezählter Primärschlüssel für den Administrator vergeben und gespeichert.

Tabelle users In diese Tabelle wird die Name eines neuen Zertifikatbesitzers eingetragen und ein von der Datenbank automatisch hochgezählter Primärschlüssel für den User vergeben und gespeichert.

Tabelle certificates Diese Tabelle hält den systemweit eindeutigen Primärschlüssel eines Zertifikats, den Primärschlüssel des Admins, der den Zertifikatsantrag bearbeitet hat, den Primärschlüssel des Requests, mit dem der Antrag in die CA gebracht wurde, den Primärschlüssel des Users, für den das Zertifikat erstellt wurde, das Zertifikat selbst, den Gültigkeitszeitraum des Zertifikats abgegrenzt durch Validitätsbeginn und Validitätsende, das Revokationspaßwort und die Rolle, die das Zertifikat im System einnimmt.

Tabelle crlEntries In dieser Tabelle werden der Primärschlüssel des revozierten Zertifikats, der CRLEntry und das Ablaufdatum des revozierten Zertifikats gespeichert.

Tabelle privKeys In dieser Tabelle werden die Private Keys der User gespeichert. Dazu werden zusätzlich der Primärschlüssel des Zertifikats, zu dem der Private Key gehört, und das Paßwort für den Private Key abgelegt.

Tabelle cards Diese Tabelle hält die Informationen zu einer Chipkarte neben dem Primärschlüssel des zugehörigen Zertifikats.

Kapitel 7

Installation des Systems

In den folgenden Abschnitten wird die Installation und Konfiguration des Systems beschrieben. Weiterhin werden die für die Kompilierung notwendigen Schritte erläutert.

7.1 Systemarchitektur

MW

Für das Gesamtsystem ergibt sich die in Abbildung 7.1 dargestellte Struktur.

7.1.1 Die Ant-Build-Files

Für das Übersetzen der Quelltexte, das Packaging, das Deployment und das Erstellen der Distributionsversionen existieren diverse Build-Files für Ant. Sie sind entsprechend der einzelnen Komponenten der FlexiTRUST-CA angelegt.

Das zentrale Build-File befindet sich in `ca/src` und wird durch das Skript `build.sh` bzw. unter Windows `build.bat` ausgewertet. Es bietet folgende elementare Tasks:

- `[db-install]`
  Konfiguriert die Datenbank. Hierbei werden alle Daten einer eventuell bereits existierenden Datenbank gelöscht!
- `[jboss-install]`
  Konfiguriert den JBoss-Application-Server.
- `[dist-single]`
  Erzeugt die Distributionsversionen der FlexiTRUST-CA. Es werden für die eigentliche CA mit dem EJB-Tier und den Serverteil je ein Archiv im Verzeichnis

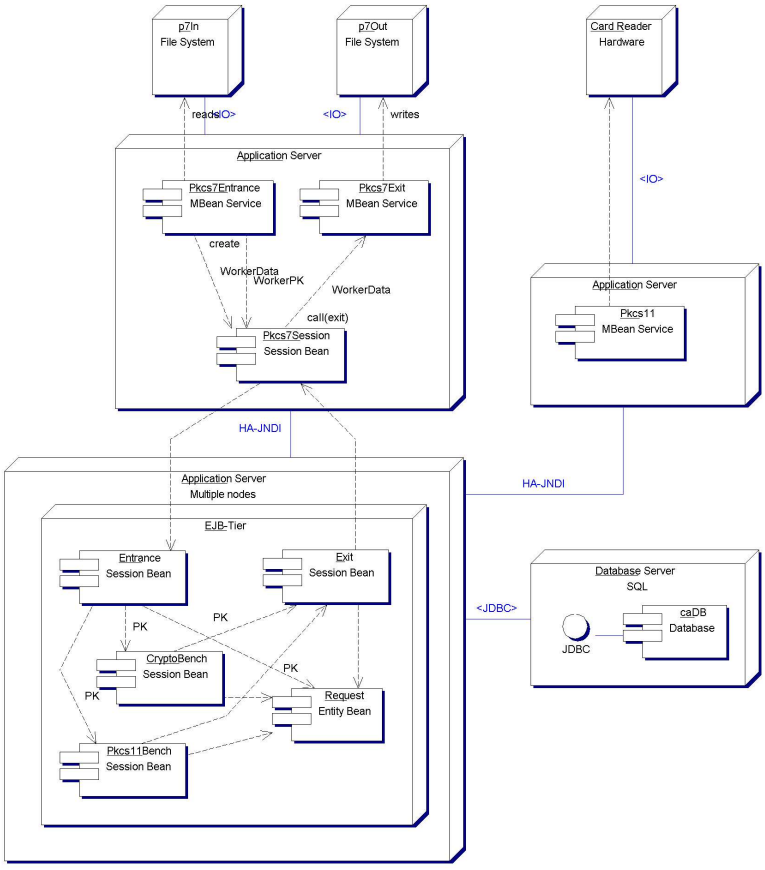

Abbildung 7.1: Systemarchitektur

`ca/dist` erzeugt, welche auf die im Cluster verwendeten Nodes übertragen werden müssen. Es können beliebige Kombinationen der EAS verwendet werden, wobei pro Node nur ein Application-Server (EAS) möglich ist. Bsp: Node 1 & 2 FlexiTRUST-Server; Nodes 3,4 & 5 FlexiTRUST-CA.

- `[dist-multi]`
  Erzeugt die Distributionsversionen der FlexiTRUST-CA für den gleichzeitigen Betrieb zweier JBoss-Instanzen pro Node. Hiermit ist es möglich, einen FlexiTRUST-Server und eine FlexiTRUST-CA auf einem Node zu starten. Dadurch wird die Simulation eines Clusters mit nur einem Node möglich. Das Hinzufügen weiterer CAs und/oder Server auf anderen Nodes ist zur Laufzeit möglich. Bsp: Node 1 FlexiTRUST-Server & FlexiTRUST-CA, Node 2 FlexiTRUST-Server, Node 3,4 & 5 FlexiTRUST-CA.

- `[xdoclet]`
  Generiert die Interfaces und Deployment-Deskriptoren der EJBs und MBeans der CA bzw. des Servers.

- `[lib-calib]`
  Erzeugt ein Jar-File mit von FlexiTRUST-CA benötigten Bibliotheken. In ihr sind die Pakete FlexiCoreProvider, Codec, AspectJRT und JWS enthalten.

- `[deploy-scratch]`
  Erzeugt die FlexiTRUST-CA und konfiguriert die Datenbank und den Application-Server. Dieses Target ist insbesondere für Testzwecke geeignet.

- `[deploy-unittests]`
  Erzeugt die Unittests.

- `[audit]`
  Erzeugt die JavaDoc und führt die Sourcecode Qualitätsüberprüfung durch.

- `[all]`
  Kombiniert die Targets `[deploy-scratch]` und `[audit]`.

- `[run]`
  Startet den JBoss-Application-Server. Hier ist jedoch zu bemerken, daß JBoss von Ant gestartet wird und damit ein Performanceverlust einhergeht.

Ergebnismatrix

Die folgende Matrix gibt Aufschluß darüber, welche Komponenten von den oben erwähnten Targets erzeugt werden.

	`[db-install]`	`[jboss-install]`	`[dist-single]`	`[dist-multi]`	`[xdoclet]`	`[lib-calib]`	`[deploy-scratch]`	`[deploy-unittests]`	`[audit]`	`[all]`	`[run]`
DB	ⓧ						ⓧ			ⓧ	
JBoss		ⓧ					ⓧ			ⓧ	
Server Dist			ⓧ	ⓧ							
CA Dist			ⓧ	ⓧ							
Interfaces			ⓧ	ⓧ	ⓧ		ⓧ	ⓧ	ⓧ	ⓧ	
CA-EJBs			ⓧ	ⓧ			ⓧ			ⓧ	
CA-Library			ⓧ	ⓧ		ⓧ	ⓧ			ⓧ	
Pkcs11Server			ⓧ	ⓧ			ⓧ			ⓧ	
Pkcs7Server			ⓧ	ⓧ			ⓧ			ⓧ	
Unittests								ⓧ	ⓧ	ⓧ	
JavaDoc										ⓧ	
CheckStyle										ⓧ	
Start JBoss											ⓧ

7.2 Datenbank Server

LF

Die dieser Diplomarbeit vorausgegangene Studienarbeit [Fel02] gab die Empfehlung für MySQL als zu verwendendes Datenbank-Management-System. Daher kommt für die persistente Speicherung der anfallenden Daten MySQL-Max ab Version 3.23.52 [Com02a] zum Einsatz. Da die Tabellen der Standardversion von MySQL keine Transaktionalität liefern, ist die Max-Version zwingend vorgegeben, um die Unterstützung von InnoDB-Tabellen, welche Transaktionalität liefern, zu gewährleisten.

MySQL bietet die Möglichkeit, eine aktive Datenbank (im folgenden Master genannt) auf anderen Datenbankservern (im folgenden Replikanten genannt) zu replizieren. Das bedeutet, daß ein oder mehrere MySQL-Server, die als Replikanten konfiguriert sind, eine Verbindung zum Master herstellen und über die Binary-Logs des Masters dessen Änderungen an der Datenbank übernehmen. Binary-Logging bedeutet, daß jede Änderung der Datenbank unmittelbar nach dem `commit` in ein binäres Log geschrieben wird.

Da die Aktualisierung über dieses Binary-Log innerhalb weniger Sekunden nach Log-Ausgabe des Masters erfolgt, ist es möglich, diese Ausgabe zu verzögern, um größere Änderungen am Stück aktualisieren zu lassen. Durch dieses Vorgehen wird sichergestellt, daß immer mindestens eine vollständige Kopie des Masters verfügbar ist. Der Einsatz von Replikanten ist nicht ausreichend für eine hoch zuverlässige Sicherung der Daten, für eine zusätzliche regelmäßige Sicherung hat der Datenbank-Administrator zu sorgen.

MySQL bietet zahlreiche Einstellungsmöglichkeiten wie die Größe der Datenbankdatei, die Anzahl der maximal zulässigen Verbindungen, Cache-Größen und Buffer-Größen; hier muß experimentell in einem realistischen Test-Betrieb die beste Konfiguration für den Betrieb mit der FlexiTRUST-CA ermittelt werden.

In dieser Installationsanleitung wird als Datenbankname `caDB` benutzt.

7.2.1 Systemvoraussetzungen

Systemvoraussetzungen der Hardware sind:

- Pentiumclass System > 500 MHz,
- 256MB Arbeitsspeicher,
- ausreichend Speicherplatz.

Systemvoraussetzungen der Software sind:

- Betriebssystem Linux (Kernel 2.4), Solaris 7 oder Microsoft Windows 2000 oder höher,
- DBMS MySQL Max 3.23.52 oder höher.

7.2.2 Installation des Datenbank Masters

Nach Installation von MySQL Max gemäß Anleitung [Com02c] (bzw. mitgelieferter `INSTALL`-Datei) ist der Datenbankserver zu deaktivieren, damit weitere Konfigurationsschritte vorgenommen werden können. Unter Windows wird dies über die System-Services durchgeführt, unter Unix genügt das als `root` ausgeführte Kommando

```
[~] /etc/init.d/mysql stop.
```

Ist der Server deaktiviert, muß im Installationsverzeichnis eine Konfigurationsdatei angelegt werden (Windowsstandard `C:\mysql\my.ini`, Unixstandard `/var/lib/mysql/my.cnf`). Eine Beispielkonfiguration mit aktiviertem Binary-Logging (siehe Abschnitt 7.2 „Replikation der Datenbank“) ist in `ca/install/mysql-max/my.cnf.master` angelegt. Diese Beispielkonfiguration erstellt eine Datenbank mit der initialen Größe von 100 Megabyte, die bei Bedarf automatisch auf eine maximale Größe von 1000 Megabyte ausgedehnt wird. Unter Windows existiert `my.ini` bereits, diese muß mit den Einträgen von `my.cnf.master` angepaßt werden.

Da der Datenbankserver keine Verzeichnisse anlegt, ist als user `mysql` das Verzeichnis `/var/lib/mysql/caDB` (für Windows `C:\mysql\data\caDB`) anzulegen. Nun kann der Server reaktiviert werden, bei Windows erfolgt dies analog zum Deaktivieren, unter Unix über das Kommando

```
[~] /etc/init.d/mysql start.
```

MySQL erzeugt nun die erforderlichen Dateien. Dieser Vorgang kann je nach konfigurierter Datenbankgröße länger dauern.

Falls die Datenbank repliziert werden soll (siehe Abschnitt 7.2.3 „Replikation der Datenbank“), muß an dieser Stelle der Datenbankserver erneut deaktiviert werden, um einen Snapshot der erzeugten Dateien zu sichern. Dafür sichert man das Verzeichnis `caDB` in einem Archiv (Windows: WinACE, WinZIP oder WinRAR, Unix: tar und/oder gzip) als user `mysql`. Beispiel:

```
[~] tar cvfz caDB initial_snapshot.tar.gz.
```

Nun kann der Server wieder gestartet werden, um die erforderlichen Benutzer anzulegen.

Für den Zugriff der CA auf die Datenbank muß ein Benutzer (momentan `flexiTrust`) mit Paßwort (momentan `flexiTrust`) mit vollen Zugriffsrechten von allen Hosts auf die Datenbank eingerichtet werden. Die hierfür erforderlichen Schritte sind dem MySQL Manual [Com02b] zu entnehmen. Dieser Benutzer ist dem Application-Server JBoss mitzuteilen. Momentan geschieht dies über die Konfigurationsdatei `$JBOSS_HOME/server/flexiTRUST/deploy/mysql-service.xml`. Der geplante Einsatz von JAAS [Inc02d] wird dies überflüssig machen und eine andere Handhabung einführen.

Zum Anlegen der erforderlichen Tabellen ist das ant-target `[db-install]` im Verzeichnis `ca/src` aufzurufen.

7.2.3 Installation eines Replikanten

Installation von MySQL Max analog zum Master (siehe 7.2.2) mit der Ausnahme, daß als Vorlage für die Konfigurationsdatei `my.cnf` die Datei `ca/install/mysql-max/-my.cnf.replicant` verwendet wird (hierbei müssen eindeutige Server-IDs vergeben werden!) und hier als Master-Host die IP-Adresse oder dessen Name - falls dieser aufgelöst werden kann - einzutragen ist.

Zusätzlich muß auf dem Master ein neuer Benutzer `replicant` mit Paßwort `replicant` eingetragen werden, der die Rechte `Select`, `Reload`, `Process`, `File` benötigt und von allen Hosts auf den Master zugreifen können muß.

Bevor der Replikant wieder reaktiviert werden kann, muß die gesicherte Snapshotdatei (siehe Abschnitt 7.2.2 „Snapshot“) im MySQL-Verzeichnis (Windows: `C:\mysql\-data`, Unix: `/var/lib/mysql`) als user `mysql` entpackt werden. Der darauf folgende Start des Replikanten initialisiert nun die Verbindung zum Master und das Update über das Master-Log.

7.2.4 Update der Anwendung

Bei einem eventuellen Update des Datenbank Management Systems sind sämtliche dokumentierte Änderungen gegenüber dem bestehenden System mit größter Sorgfalt zu überprüfen und es ist sicherzustellen, daß die bestehende Datenbank vollständig und ohne Nebeneffekte übernommen werden kann, andernfalls ist mit Datenverlust zu rechnen. Eine Überprüfung auf Veränderung der Transaktionseigenschaften und Einhaltung der Referenzbedingungen des verwendeten Tabellentyps InnoDB ist sehr zu empfehlen, da ansonsten eine unkorrekte Speicherung im laufenden Betrieb dazu führen kann, daß die Integrität der gespeicherten Daten verletzt wird.

7.2.5 Deinstallation

Falls das Datenbank Management System durch ein anderes ersetzt werden soll, ist in der Anleitung des zukünftigen Systems nachzulesen, wie die Inhalte von MySQL InnoDB Tabellen importiert werden können und wie die Datei `ca/personal-server/-db/database.sql` an das neue System angepaßt werden kann.

Die Deinstallation von MySQL kann dann so erfolgen, wie Deinstallationen auf dem verwendeten System üblich sind.

7.3 Enterprise Application Server

MW

Als Plattform der FlexiTRUST-CA wird JBoss Version 3.0.3 [Gro02b] als Enterprise-Application-Server (EAS) eingesetzt. Für eine nähere Betrachtung dieses EAS sei auf die Studienarbeit [Fel02] verwiesen.

7.3.1 Systemvoraussetzungen

Systemvoraussetzungen der Hardware sind:

- Pentiumclass System > 500 MHz,
- 256 MB Arbeitsspeicher,
- 500 MB HD.

Systemvoraussetzungen der Software sind:

- Betriebssystem Linux (Kernel 2.4), Solaris 7 oder Microsoft Windows 2000 oder höher,
- Application-Server: JBoss 3.0.3[1] oder höher,
- J2SDK 1.4.1_01,
- Ant 1.5.

7.3.2 Vorbereitende Schritte

Installation von JBoss Application-Server

Die Installation muß gemäß [JG02] „JBoss 3.0 getting started“ Kapitel 2 erfolgen.

7.3.3 Installation der Anwendung

Environmentvariablen

Folgende Environmentvariablen müssen gesetzt sein:

[1] Der Application-Server JBoss hat in den Versionen 3.0.x einen Fehler, welcher die in diesem Projekt verwendeten verteilten Transaktionen verhindert. Für die Version 3.0.3 befindet sich im Verzeichnis `ca/personal-server` ein Patch, der dieses Problem behebt. Dieser Patch wird bei der Übersetzung der Quelltexte automatisch angewendet. Aus diesem Grund ist die FlexiTRUST-CA vorläufig nur mit Version 3.0.3 lauffähig.

- `HOME` (Bsp.: `/home/username`),
- `JAVA_HOME` (Bsp.: `/usr/local/bin/java`),
- `JBOSS_HOME` (Bsp.: `~/JBoss`),
- `FLEXITRUST_HOME` (Bsp.: `~/FlexiPKI/FlexiTRUST`).

Konfiguration der FlexiTRUST-CA

Für die Konfiguration der FlexiTRUST-CA sind folgende Dateien unterhalb `$FLEXITRUST_HOME` relevant:

Verzeichnis	Datei
`ca/config-props`	`ca.properties`
	`jndi.properties`
`ca/personal-props`	`ant.properties`
	`ca-multi.properties`
	`ca-single.properties`
	`server-multi.properties`
	`server-single.properties`

Vom FlexiTRUST-CA-Administrator sind insbesondere die Dateien `ant.properties` und `ca.properties` anzupassen. Über die restlichen Konfigurationsdateien werden JBoss-interne Parameter definiert, die normalerweise nicht angepaßt werden müssen.

Die Konfigurationsdatei ant.properties

Die Konfigurationsdatei `$FLEXITRUST_HOME/ca/personal-props/ant.properties` legt Parameter für den Übersetzungvorgang der FlexiTRUST-CA fest.

Listing 7.1: ant.properties

```
1  all.home = all
2  ca.home = ca
3  jws.home = jws
4  appserver.deploy = server/flexiTRUST/deploy
5
6  ajc.use=defined
7  ajc.list=develop.lst
8  aj.usejavac=false
9
10 ejtools.use=defined
11
12 #
13 # deploy/cluster-service.xml
14 #
15 # _PARTITION_
16 partition.name=FlexiTRUST
17
18 #
19 # deploy/mysql-service.xml
20 #
```

```
21 # _MYSQLIP_ (134.101.26.85)
22 mysql.ip = <MySQL-Server-IP>
23
24 # _MYSQLPORT_ (3306)
25 mysql.port = <MySQL-Server-Port>
26
27 #
28 # JBoss-Version (3.0.3)
29 #
30 jboss.version = <JBoss-Version>
```

Zeile 1 ff.: Speicherorte der FlexiTRUST-Komponenten `all`, `ca` und `jws` relativ zu `$FLEXITRUST_HOME`.

Zeile 4: Verzeichnis der FlexiTRUST-Konfiguration für JBoss relativ zu `$JBOSS_HOME`.

Zeile 6 ff.: Verwendung des AspectJ-Compilers. Falls diese Property gesetzt wird, muß auch `ajc.list` einen gültigen Wert definieren.

Zeile 10: Einbinden des erweiterten JMX-Frontends der EJTools zusätzlich zu der JBoss-eigenen JMX-Console. Ist dieses nicht gewünscht, genügt ein einfaches Auskommentieren der Zeile.

Zeile 16: Definiert den Namen der Clusterpartition. Durch Vergabe unterschiedlicher Partitionsnamen wird es möglich, in einem LAN mehrere FlexiTRUST-CA-Cluster in Betrieb zu nehmen, die unabhängig voneinander arbeiten.

Zeile 22 ff.: Definiert die IP-Adresse und den Port des zu verwendenden MySQL-Datenbank-Servers.

Zeile 30 ff.: Definiert die verwendete JBoss-Version[2].

Die Konfigurationsdatei ca.properties

Die Konfigurationsdatei `$FLEXITRUST_HOME/ca/config-props/ca.properties` legt Parameter für die Laufzeit der FlexiTRUST-CA fest.

Listing 7.2: ca.properties

```
1  #
2  # Pkcs7Server
3  #
4  pkcs7Server.p7InDirName = ../../FlexiPKI/FlexiTRUST/p7In
5  pkcs7Server.p7OutDirName = ../../FlexiPKI/FlexiTRUST/p7Out
6  pkcs7Server.p7ErrDirName = ../../FlexiPKI/FlexiTRUST/p7Err
7
8  #
9  # Pkcs7EntranceService
10 #
11 pkcs7EntranceService.threadPoolSize = 5
12 pkcs7EntranceService.filePollingInterval = 3000
13
14 #
```

[2] Aufgrund eines notwendigen Patches des JBoss-Application-Servers sollte im momentanen Entwicklungsstand der FlexiTRUST-CA hier die Versionsnummer 3.0.3 eingetragen werden.

```
15 # Pkcs7ExitService
16 #
17 pkcs7ExitService.maxProcessingDelay = 10000
18 pkcs7ExitService.maxProcessables = 33
19 pkcs7ExitService.statPulseLimit = 60000
20 pkcs7ExitService.idleTime = 333
21 pkcs7ExitService.outFileExt = p7
22 pkcs7ExitService.outFileName = caOut
23 pkcs7ExitService.statFileName = p7ExitStat.csv
24 pkcs7ExitService.statDirName = ../../FlexiPKI/FlexiTRUST/p7Stat
25
26 #
27 # UnitTest
28 #
29 unittest.datDirName = ../../FlexiPKI/FlexiTRUST/ca/tst/dat
```

Zeile 4 ff.: Verzeichnisse `p7In`, `p7Out` und `p7Err` relativ zu `$JBOSS_HOME/bin`.

Zeile 11 ff.: Konfiguration des Pkcs7EntranceService. Es können die `ThreadPoolSize` und das `FilePollingInterval` angegeben werden.

Zeile 17 ff.: Konfiguration des Pkcs7ExitService. Es können die Attribute `MaxProcessingDelay`, `Maxworkers`, `StatPulseLimit`, `IdleTime`, `OutFileExt`, `OutFileName`, `StatFileName` und `StatDirName` angegeben werden. Ein `StatPulseLimit=0` deaktiviert die Statistik.

Zeile 29 ff.: KeyStore-Verzeichnis der Unittests relativ zu `$JBOSS_HOME/bin`.

Sourcecodeversion der FlexiTRUST-CA

Die Quelltexte der FlexiTRUST-CA befinden sich im CVS-Repository

```
          [~]$ cd $FLEXITRUST_HOME
[FlexiTRUST]$ export CVSROOT=":ext:<username>@
              <cvs-server>:<cvs-directory>"
[FlexiTRUST]$ export CVS_RSH="ssh"
[FlexiTRUST]$ cvs co -r ejb all ca jws
```

Übersetzt und konfiguriert wird die FlexiTRUST-CA durch ein

```
[src]$ ./build.sh all
```

im Verzeichnis `ca/src`.

7.3.4 Start der FlexiTRUST-CA

Der Application-Server mit FlexiTRUST-Konfiguration wird über den Aufruf eines Startskripts im Verzeichnis `$JBOSS_HOME/bin` gestartet.

```
[bin]$ ./startFlexiTRUST.sh
```

7.3.5 Initialisierung der FlexiTRUST-CA

Für die Inbetriebnahme der CA müssen ein initialer Issuer und ein initialer Administrator erstellt werden. Im momentanen Entwicklungsstand ist dieses durch den Unittest `TestInitKeyRequest` realisert. Für den Issuer wird zunächst ein KeyPair erzeugt und ein mit dem Private-Key signiertes Zertifikat erstellt. Dieses wird mit einem IssuerRequest über Funktionalitäten der Entrance direkt in den Stock der CA eingebracht. Zusätzlich werden die erzeugten Daten für weitere Verfügbarkeit in einen KeyStore im Datenverzeichnis der Unittests geschrieben.

Anschließend wird ein KeyPair und ein Zertifikat für einen Administrator erstellt und mit dem Private-Key des initialen Issuers signiert. Diese Daten werden über einen AdminRequest analog zum initialen Issuer über eine Entrance-Funktionalität in den Stock eingebracht. Auch die Daten des Administrators werden in einen KeyStore im Datenverzeichnis der Unittests geschrieben, um über den PrivateKey des Administrators für die Signierung von Requests verfügen zu können.

Nach Abschluß dieses Requests ist die Initialisierung abgeschlossen und die FlexiTRUST-CA betriebsbereit.

7.3.6 Herunterfahren der FlexiTRUST-CA

Der Application-Server kann mit Hilfe des Skriptes `shutdown.sh` herunter gefahren werden.

```
[bin]$ ./shutdown.sh
```

7.3.7 Update der Anwendung

Update von JBoss

Ein Update des Application-Servers wird analog zur Installation (vgl. Abschnitt 7.3.2 „Installation von JBoss“) durchgeführt.

Anschließend muß die FlexiTRUST-CA neu übersetzt werden.

Update der FlexiTRUST-CA

Das Update der CA erfolgt durch ein CVS-Update der Module `all`, `ca` und `jws`.

Anschließend müssen die aktualisierten Packages neu übersetzt werden.

7.3.8 Deinstallation

Die Deinstallation erfolgt durch Löschen der Verzeichnisse `$JBOSS_HOME` und `$FLEXITRUST_HOME`.

7.3.9 Distributionsversion der FlexiTRUST-CA

Eine Distribution der FlexiTRUST-CA enthält, voneinander getrennt, ein CA-EJB-Tier und einen Pkcs-Serverteil. Diese können über

```
[src]$ ./build.sh dist-single
```

oder

```
[src]$ ./build.sh dist-multi
```

im Verzeichnis `$FLEXITRUST_HOME/ca/src` erzeugt werden und sind nach Fertigstellung in Tar-Archiven im Verzeichnis `$FLEXITRUST_HOME/ca/dist` zu finden. Sie beinhalten vollständig für einen Cluster konfigurierte JBoss-Server. Vorbedingung hierfür ist eine frische JBoss-Installation im Verzeichnis `$JBOSS_HOME`. Das Archiv `FlexiTRUST-Server.tar.gz` beinhaltet die Komponente `Pkcs7Server` und stellt somit den Ein- und Ausgang der CA dar. Das Archiv `FlexiTRUST-CA.tar.gz` beinhaltet das EJB-Tier der CA mit den Requests, den Workbenches und dem Stock.

Bei einer mit `[dist-multi]` erzeugten Distribution können je Node ein FlexiTRUST-Server und eine FlexiTRUST-CA gestartet werden. Diese eignen sich daher auch für die Simulation eines Clusters mit nur einem Computer (vgl. auch Abschnitt 7.1.1 „Die Ant Build Files“). Ist eine Distribution mit `[dist-single]` erstellt worden, ist nur die Verwendung jeweils eines Application-Servers möglich.

Die erzeugten Archive können nun auf die einzelnen Nodes des Clusters übertragen werden. Nach einem Entpacken über

```
[~]$ tar xfvz FlexiTRUST-<NAME>.tar.gz
```

können die Application-Server mit

```
  [~]$ cd FlexiTRUST-<NAME>/bin
[bin]$ startFlexiTRUST.sh
```

gestartet werden. Nach dem Startvorgang und dem automatischen Einbinden in den Cluster sind die EAS einsatzbereit. Das Hinzufügen weiterer CAs und/oder Server auf anderen Nodes ist zur Laufzeit möglich.

Die Erzeugung einer Distributionsversion schließt die Unittests nicht mit ein.

Kapitel 8

Betrieb des Systems

Für die Überwachung des Systems werden umfangreiche Log-Files erzeugt. Einzelne Komponenten bieten eine Laufzeitüberwachung über das JMX-Frontend.

8.1 Datenbank Server

LF

MySQL erzeugt im laufenden Betrieb eine Datei `<hostname>.err`. Die Dateiendung läßt zwar vermuten, daß hier nur Fehler protokolliert werden, die Datei wird aber für alle von MySQL erzeugten Meldungen verwendet.

8.2 Application Server

MW

Zur Laufzeit der FlexiTRUST-CA werden im Verzeichnis

```
$JBOSS_HOME/server/flexiTRUST/log
```

Log-Files der einzelnen Komponenten erzeugt. Dies sind im Einzelnen:

- `boot.log`: Meldungen während des Systemstarts,
- `cluster.log`: Meldungen des Cluster-Systems,
- `flexitrust.log`: Meldungen der FlexiTRUST-CA,
- `server.log`: Meldungen des Application-Servers zur Laufzeit.

8.3 Pkcs7Server

MW

Der Pkcs7Server besteht aus zwei MBeans, die Services für die FlexiTRUST-CA bereit stellen. Dies ist zum einen der Pkcs7Entrance, der die signierten Requests aus dem Verzeichnis `p7In` entnimmt, verifiziert und an das EJB-Tier weitergibt. Zum anderen der Pkcs7Exit, der die bearbeiteten Requests entgegennimmt, signiert und in das `p7Out`-Verzeichnis schreibt. Beide Services können entweder über die Konfigurationsdatei `ca.properties` oder zur Laufzeit über das JMX-Frontend konfiguriert werden.

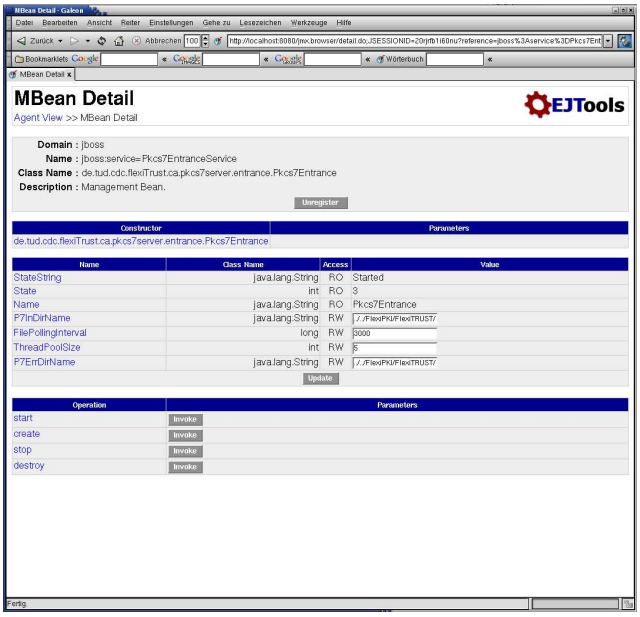

Abbildung 8.1: MBean Detail Pkcs7Entrance

Um den Pkcs7Entrance zur Laufzeit zu konfigurieren sind folgende Schritte notwendig:

1. Mit einem Browser die Adresse `http://<IP-Adresse-des-FlexiTRUST-Servers>:8080/jmx.browser` öffnen.
2. In der Kategorie „`jboss`“ den „`service=Pkcs7EntranceService`“ auswählen (siehe Abbildung 4.9, welche die Detailseite des Services innerhalb des JMX-Browsers der EJTools darstellt).
3. In der Liste der Attribute die wie gewünscht ändern und den Button `[Update]` betätigen (siehe auch Abbildung 8.1).

Es ist darauf zu achten, daß diese Änderung nur bei Inaktivität des EntranceService durchgeführt wird, da die in Arbeit befindlichen Requests ansonsten abgebrochen und mit dem Status `failed` in der Datenbank markiert werden.

Für die Konfiguration des Pkcs7Exit muß bei den JBoss-Services der Service `Pkcs7-ExitService` ausgewählt werden (siehe Abbildung 8.2).

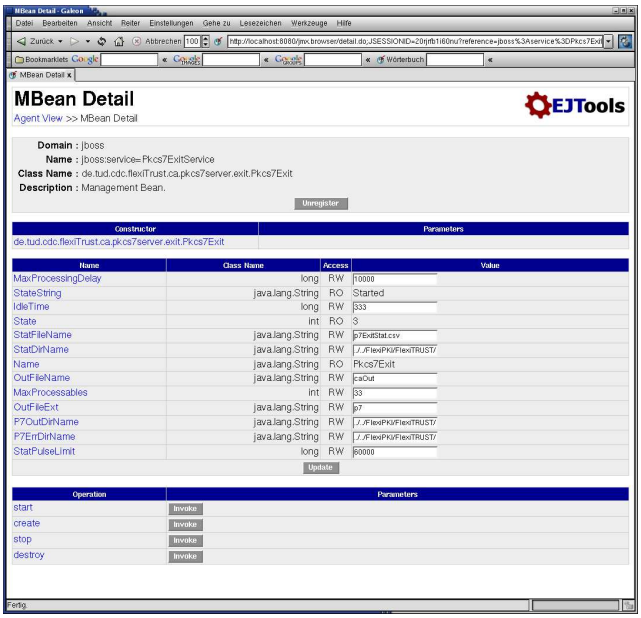

Abbildung 8.2: MBean Detail Pkcs7Exit

Kapitel 9

Erweiterung des Systems

MW+LF

Durch die modulare Struktur des Systems wird es möglich, neue Funktionalitäten einzubringen. Die dafür notwendigen Schritte werden in der folgenden beispielhaften Implementierung eines X509v3 Certification-Request mit PKCS#11[1] erläutert. Dieser Request soll den Zugriff auf eine SmartCard über einen SmartCardReader realisieren und somit die Personalisierung einer SmartCard ermöglichen.

Für diese Erweiterung ist es notwendig, eine Workbench und einen Request zu implementieren. Für den Hardwarezugriff auf die SmartCards wird ein MBean-Service vorgesehen.

9.1 Spezifikation der Anwendungsfälle und Klassen

In den folgenden Abschnitten werden die durch eine Erweiterung des Systems hinzukommenden Anwendungsfälle und Klassen beschrieben. Sie stellen somit eine Ergänzung zu Abschnitt 3.4 „Spezifikation der Anwendungsfälle“ dar. Es werden hier nur die minimal notwendigen Fälle betrachtet, eine umfassende Spezifikation für alle notwendigen Funktionalitäten obliegt den Erweiterungen durchführenden Systementwicklern. In den Abbildungen werden nur die hinzugekommenen oder erweiterten Anwendungsfälle und Klassen dargestellt.

[1] Pkcs11 bezieht sich auf den Cryptographic Token Interface Standard. Vergleiche hierzu [Lab02].

9.1.1 Spezifikation der Anwendungsfälle

Anwendungsfälle der Domain Bench

Die nachfolgende Abbildung 9.1 zeigt alle Anwendungsfälle des Bereichs `Bench` und ihre Beziehungen untereinander.

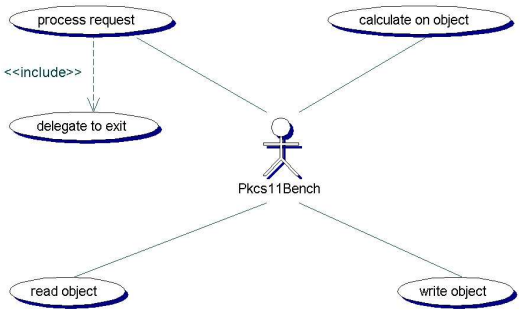

Abbildung 9.1: Anwendungsfälle der Domain Bench

- `process request`: Dieser Anwendungsfall beschreibt die Abarbeitung von Arbeitsschritten des Requests auf der Bench. Hier wird eine lokale Instanz des Requests erzeugt und dessen Methode `work` aufgerufen. Die Bench stellt ihm hierbei ihre spezifischen Funktionalitäten zur Verfügung.
- `delegate to exit`: Nachdem ein Request alle auf dieser Bench möglichen Arbeitsschritte abgeschlossen hat, wird er von dieser an die von ihm spezifizierte nächste Bench weitergegeben.
- `read object`
  `write object`
  `calculate on object`: Die Pkcs11Bench stellt den Requests Methoden für den Zugriff auf die Kartenhardware zur Verfügung.

Anwendungsfälle der Domain Request

Die nachfolgende Abbildung 9.2 zeigt alle Anwendungsfälle des Bereichs `Request` und ihre Beziehungen untereinander.

- `work p11`: Die Spezialisierung des Anwendungsfalls `work` für den `X509P11Request`.

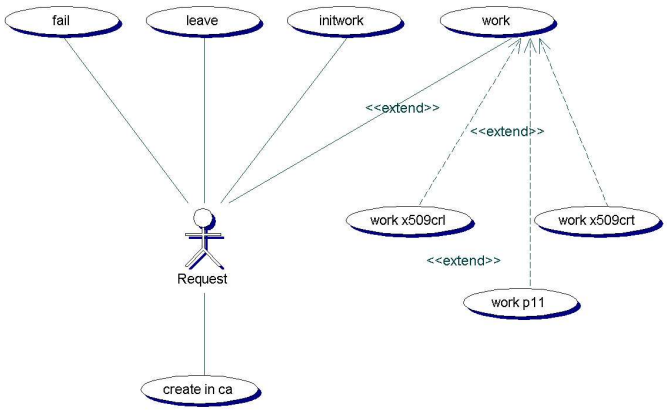

Abbildung 9.2: Anwendungsfälle der Domain Request

- Die weiteren Anwendungsfälle wurden bereits in Abschnitt 3.4 „Anwendungsfälle der Domain Request“ beschrieben.

Anwendungsfälle des Pkcs11Server

Der `Pkcs11Server` untergliedert sich in die zwei Komponenten `CardAccess` und `Session`, deren Anwendungsfälle in den folgenden Abschnitten beschrieben werden.

Anwendungsfälle der Domain CardAccess Die nachfolgende Abbildung 9.3 zeigt alle Anwendungsfälle der Komponente `CardAccess` des `Pkcs11Servers` und ihre Beziehungen untereinander.

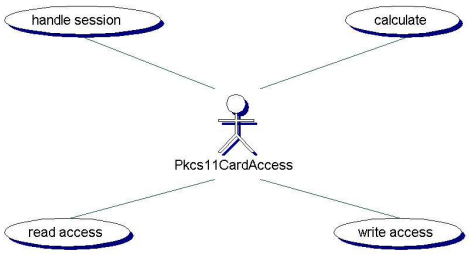

Abbildung 9.3: Anwendungsfälle des Pkcs11Server CardAccess

Die Anwendungsfälle des `CardAccess` beschreiben vom System durchzuführende Anwendungsfälle für den Zugriff auf die SmartCard-Hardware.

- `handle session`: Der `Pkcs11CardAccess` stellt ein Sessionhandling für den SmartCard-Zugriff zur Verfügung.
- `read access`
  `write access`: In diesen Anwendungsfällen stellt der `CardAccess` Funktionalitäten für den lesenden und schreibenden Zugriff auf die SmartCards zur Verfügung.
- `calculate`: In diesem Fall werden Kalkulationen auf der SmartCard ausgeführt.

Anwendungsfälle der Domain Session Die nachfolgende Abbildung 9.4 zeigt alle Anwendungsfälle der Komponente `Session` des `Pkcs11Servers` und ihre Beziehungen untereinander.

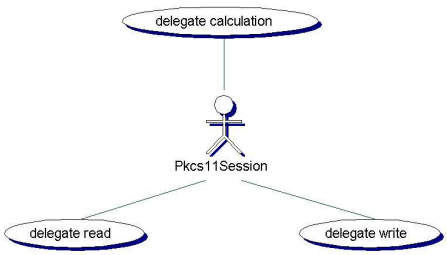

Abbildung 9.4: Anwendungsfälle des Pkcs11Server Session

Sie beschreiben vom System durchzuführende Anwendungsfälle bezüglich der Weiterleitung von Anfragen seitens des EJB-Tiers an die Komponente CardAccess.

- `delegate read`: Die `Pkcs11Session` delegiert den lesenden Zugriff auf die SmartCard an den `CardAccess` und liefert das Ergebnis zurück.
- `delegate write`: Die `Pkcs11Session` delegiert den schreibenden Zugriff auf die SmartCard an den `CardAccess`.
- `delegate calculation`: Die `Pkcs11Session` delegiert eine Kalkulation an den `CardAccess` und liefert das Ergebnis zurück.

9.1.2 Identifikation der Geschäftsklassen

Die mit Hilfe der Anwendungsfälle erkannten Geschäftsklassen werden in den folgenden Klassendiagrammen dargestellt. Die Darstellung erfolgt in UML-Notation. Die

Menge aller Geschäftsklassen bildet die Grundlage des Design-Modells, d.h. die Darstellung entspricht nicht mehr der Analysesicht und berücksichtigt bereits Implementierungsaspekte (vgl. hierzu [Win02]).

Geschäftsklassen der Domain Bench

Im Paket `Bench` sind die zentralen Klassen der Workbenches mit den notwendigen Funktionalitäten zur Requestbearbeitung zusammengefaßt.

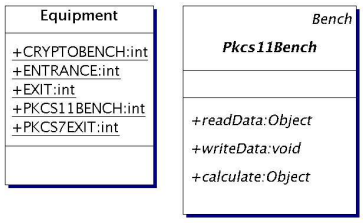

Abbildung 9.5: Geschäftsklassen der Domain Bench

Die in Abbildung 9.5 dargestellten Klassen werden in dem nachfolgenden Abschnitten beschrieben.

Klasse Pkcs11Bench Die Klasse `Pkcs11Bench` stellt Funktionalitäten für die Bearbeitung von SmartCards wie Lesen und Schreiben der Karten zur Verfügung. Weiterhin werden Methoden für die Durchführung von Berechnungen auf den SmartCards bereitgestellt. Diese Funktionalitäten können direkt von den Requests in Anspruch genommen werden.

Klasse Equipment Die Klasse `Equipment` beinhaltet eine Auflistung aller in der CA implementierten Benches. Im Vergleich zu Abschnitt 3.6 „Geschäftsklassen der Domain Bench“ ist sie um die `Pkcs11Bench` erweitert.

Geschäftsklassen der Domain Request

Im Paket `Request` sind die zentralen Klassen der Requests zusammengefaßt.

Die in Abbildung 9.6 dargestellten Klassen werden in dem nachfolgenden Abschnitten beschrieben.

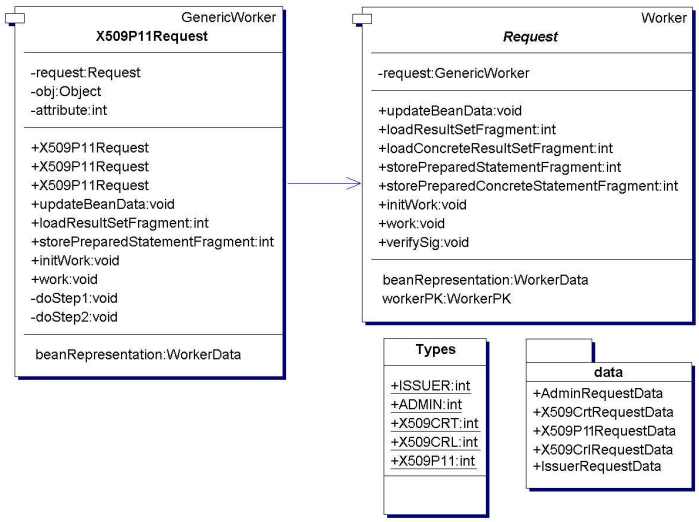

Abbildung 9.6: Geschäftsklassen der Domain Request

Request Die Klasse `Request` beschreibt einen Request innerhalb der FlexiTRUST-CA. Neben den beschreibenden Attributen beinhaltet sie einen Verweis auf einen der spezialisierten Requests. Der `Request` bietet unterstützende Funktionalitäten für die persistente Datenspeicherung und stellt grundlegende, für alle Requests notwendige Methoden zur Verfügung. Diese sind, neben der Delegierung der Verifikation der Requestsignatur an die jeweilige Bench, für die Aktualisierung der Attribute verantwortlich. Seine Hauptaufgabe besteht jedoch in der Delegierung der Arbeitsmethoden zu den im Anschluß folgenden spezialisierten Requests.

X509P11Request Die Klasse `X509P11Request` beschreibt einen X509v3 Certification-Request und erzeugt ein PKCS#11 Ergebnis.

Types Die Klasse `Types` beinhaltet eine Auflistung aller in der CA implementierten Requesttypen. Im Vergleich zu Abschnitt 3.9 „Geschäftsklassen der Domain Request“ ist sie um den `X509P11Request` erweitert.

9.2 Systemarchitektur

Die nachfolgenden Abschnitte beschreiben die neuen und erweiterten Komponenten der FlexiTRUST-CA und ihre Aufgaben entsprechend ihrer Zuordnung zu den ver-

schiedenen Schichten der J2EE-Architektur.

9.2.1 Architektur EJB-Tier

Bench

Im Paket `Bench` sind die Enterprise JavaBeans der Workbenches zusammengefaßt. In Abbildung 9.7 ist die Session-Bean `Pkcs11BenchEJB` dargestellt.

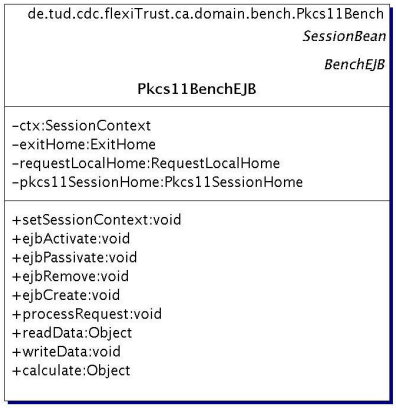

Abbildung 9.7: Enterprise JavaBeans Bench

Die in Abbildung 9.7 dargestellte Klasse wird in dem nachfolgenden Abschnitt beschrieben.

Klasse Pkcs11BenchEJB Die Stateless-Session-Bean `Pkcs11BenchEJB` beinhaltet alle bislang implementierten Operationen, die für den Kryptographiestandard Pkcs#11 in Verbindung mit weiteren Enterprise-JavaBeans notwendig sind. Ihre wesentliche Aufgabe ist die Delegierung an den Pkcs11Server, der den Hardwarezugriff realisieren wird.

9.2.2 Architektur Pkcs11Server

Der `Pkcs11Server` implementiert den Hardwarezugriff auf die SmartCards. Er untergliedert sich in die Komponenten `CardAccess` und `Session`, die im folgenden beschrieben werden.

CardAccess

Im Paket `CardAccess` sind die MBean-Service-Klassen des `Pkcs11CardAccess` und die dazugehörigen Hilfsklassen zusammengefaßt. In Abbildung 9.8 ist die MBean `Pkcs11CardAccess` dargestellt.

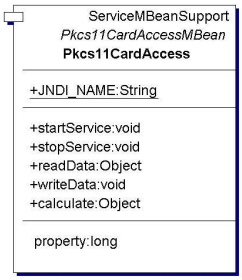

Abbildung 9.8: MBean Service Pkcs11CardAccess

Die in Abbildung 9.8 dargestellte Klasse wird in dem nachfolgenden Abschnitt beschrieben.

Klasse Pkcs11CardAccess Die Klasse `Pkcs7Exit` implementiert eine Managed-Bean und dient der Konfiguration der noch zu implementierenden Methoden und Klassen für den Hardwarezugriff.

Session

Im Paket `Session` sind die Enterprise-JavaBean der `Pkcs11Session` und die dazugehörigen Hilfsklassen zusammengefaßt. In Abbildung 9.9 ist die Enterprise-JavaBean `Pkcs11SessionEJB` dargestellt, die in dem nachfolgenden Abschnitt beschrieben wird.

Klasse Pkcs11SessionEJB Die Klasse `Pkcs11SessionEJB` stellt die Schnittstelle zwischen dem EJB-Tier der FlexiTRUST-CA und dem `Pkcs11Server` dar.

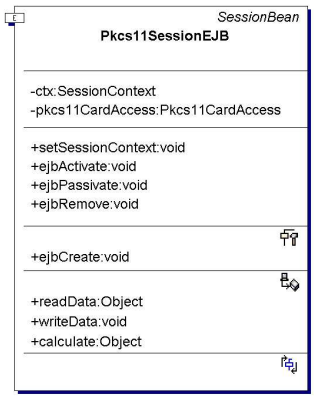

Abbildung 9.9: Enterprise JavaBeans Pkcs11Session

9.3 PKCS#11 Workbench

Beteiligte Komponenten:

Klasse	Beschreibung
`Pkcs11BenchEJB`	Beanklasse der Pkcs11Bench
`Pkcs11Bench`	Geschäftsklasse der Pkcs11Bench
`Equipment`	Auflistung aller in der CA vorhandenen Workbenches
`EntranceEJB`	Beanklasse der Entrance

Die nachfolgenden Abschnitte beschreiben die innerhalb des EJB-Tiers beteiligten Komponenten zur Realisierung von Pkcs11-Funktionalitäten und ihre Aufgaben entsprechend ihrer Zuordnung zu den verschiedenen Schichten der J2EE-Architektur.

9.3.1 Die Klasse Pkcs11BenchEJB

Die Klasse `ejbs.bench.Pkcs11BenchEJB` realisiert eine Workbench für Pkcs11-Funktionalitäten. Es findet eine Trennung zwischen dem Containerspezifischen Teil und der Geschäftslogik statt. Die Deploymentkonfiguration für den Application-Server wird über XDoclet-Tags (siehe Zeilen 22 ff.) umgesetzt.

Aus dem Tag `@ejb.bean` wird ersichtlich, daß die `Pkcs11BenchEJB` als Stateless-Session-Bean realisiert ist, wobei die Transaktionsverwaltung dem Container überlassen wird. Das Tag `@ejb.transaction` mit Typ `required` sagt aus, daß sich alle Methodenaufrufe über die EJB-Interfaces innerhalb eines Transaction-Context befinden müssen. Die Bean wird in den HA-JNDI-Baum als `ejb/CDC.FlexiTRUST.Pkcs11-BenchEJB` eingetragen. Über die Tags `@ejb.home` und `@ejb.interface` werden ihre Home- und Remote-Interfaces definiert, wobei diese Interfaces innerhalb des Build-Prozesses automatisch generiert werden. Die Tags `@ejb.ejb-ref` definieren Referenzen auf andere Beans, die innerhalb der `Pkcs11Bench` benötigt werden.

Die Tags `@jboss.clustered` und `@jboss.cluster-config` legen fest, daß diese Bean sich innerhalb der angegebenen Clusterpartition befindet. Der String `_PARTITION_` wird während des Build-Prozesses durch den in den `ant.properties` angegebene Namen ersetzt (vgl. hierzu Kapitel 7.3.3 „Installation der Anwendung“).

Listing 9.1: domain.bench.Pkcs11BenchEJB

```
1  package de.tud.cdc.flexiTrust.ca.ejbs.bench;
2
3  import java.rmi.RemoteException;
4  import javax.ejb.SessionBean;
5  import javax.ejb.SessionContext;
6  import javax.ejb.EJBException;
7  import javax.ejb.CreateException;
8  import javax.ejb.FinderException;
9  import javax.naming.NamingException;
10
11 import de.tud.cdc.flexiTrust.ca.domain.bench.Equipment;
12 import jws.ejbs.bench.BenchEJB;
13 import jws.ejbs.worker.WorkerPK;
14 import jws.util.EJBServiceLocator;
15
16 import de.tud.cdc.flexiTrust.ca.domain.bench.CaException;
17 import de.tud.cdc.flexiTrust.ca.interfaces.exit.ExitHome;
18 import de.tud.cdc.flexiTrust.ca.interfaces.request.RequestLocal;
19 import de.tud.cdc.flexiTrust.ca.interfaces.request.RequestLocalHome;
20 import de.tud.cdc.flexiTrust.ca.pkcs11server.session.Pkcs11SessionHome;
21
22 /**
23  * @ejb.bean
24  *    display-name="Pkcs11BenchEJB"
25  *    name="ejb/Pkcs11BenchEJB"
26  *    jndi-name="ejb/CDC.FlexiTRUST.Pkcs11BenchEJB"
27  *    type="Stateless"
28  *    transaction-type="Container"
29  *    reentrant="false"
30  *    use-soft-locking="true"
31  *    view-type="remote"
32  *
33  * @ejb.ejb-ref
34  *    ejb-name="ejb/RequestEJB"
35  *    view-type="local"
36  *    ref-name="ejb/RequestLocalEJB"
37  * @ejb.ejb-ref
38  *    ejb-name="ejb/ExitEJB"
39  * @ejb.ejb-external-ref
40  *    ejb-name="ejb/Pkcs11SessionEJB"
41  *    view-type="remote"
42  *    ref-name="ejb/Pkcs11SessionEJB"
43  *    type="Session"
44  *    home="de.tud.cdc.flexiTrust.ca.pkcs11server.session.
       Pkcs11SessionHome"
```

```
45   *    remote="de.tud.cdc.flexiTrust.ca.pkcs11server.session.
       Pkcs11Session"
46   *
47   * @jboss.ejb-ref-jndi
48   *    ref-name="Pkcs11SessionEJB"
49   *    jndi-name="jnp://localhost:_HAJNDIPORT_/ca-pkcs11server.ear/ejb/
       CDC.FlexiTRUST.Pkcs11SessionEJB"
50   *
51   * @ejb.home
52   *    remote-class="de.tud.cdc.flexiTrust.ca.interfaces.bench.
       Pkcs11BenchHome"
53   *
54   * @ejb.interface
55   *    remote-class="de.tud.cdc.flexiTrust.ca.interfaces.bench.
       Pkcs11Bench"
56   *
57   * @ejb.transaction
58   *    type="Required"
59   *
60   * @jboss.clustered
61   *
62   * @jboss.cluster-config
63   *    partition-name="_PARTITION_"
64   *    home-policy="org.jboss.ha.framework.interfaces.RandomRobin"
65   *    bean-policy="org.jboss.ha.framework.interfaces.RandomRobin"
66   */
67  public class Pkcs11BenchEJB
68      extends de.tud.cdc.flexiTrust.ca.domain.bench.Pkcs11Bench
69      implements SessionBean, BenchEJB {
70
71      /**
72       * The SessionContext.
73       */
74      private SessionContext ctx;
75
76      /**
77       * The ExitHome
78       */
79      private ExitHome exitHome;
80
81      /**
82       * The RequestHome.
83       */
84      private RequestLocalHome requestLocalHome;
85
86      /**
87       * The Pkcs11SessionHome.
88       */
89      private Pkcs11SessionHome pkcs11SessionHome;
90
91      /**
92       * Set the SessionContext. The Container calls this method after
93       * creation of the instance.
94       * @param context The SessionContext for this instance.
95       */
96      public void setSessionContext(SessionContext context) {
97
98          ctx = context;
99      }
100
101     /**
102      * ejbActivate.
103      */
104     public void ejbActivate() {  }
105
106     /**
107      * ejbPassivate
108      */
109     public void ejbPassivate() { }
110
111     /**
112      * ejbRemove
```

```
113         */
114     public void ejbRemove() {
115 
116         exitHome = null;
117         requestLocalHome = null;
118         pkcs11SessionHome = null;
119         ctx = null;
120     }
121 
122     /**
123      * ejbCreate
124      * @throws CreateException An entity EJB object could not be
125      * created.
126      */
127     public void ejbCreate()
128         throws CreateException {
129 
130         try {
131 
132             exitHome  =
133                 (ExitHome) EJBServiceLocator.lookupHome
134                 (ExitHome.class,
135                  "ejb/CDC.FlexiTRUST.ExitEJB");
136         } catch (NamingException ex) {
137             throw new EJBException(ex);
138         }
139 
140         try {
141 
142             requestLocalHome  =
143                 (RequestLocalHome) EJBServiceLocator.lookupLocalHome
144                 (RequestLocalHome.class,
145                  "ejb/CDC.FlexiTRUST.RequestLocalEJB");
146         } catch (NamingException ex) {
147             throw new EJBException(ex);
148         }
149 
150         try {
151 
152             pkcs11SessionHome =
153                 (Pkcs11SessionHome) EJBServiceLocator.lookupHome
154                 (Pkcs11SessionHome.class,
155                  "ejb/CDC.FlexiTRUST.Pkcs11SessionEJB");
156         } catch (NamingException ex) {
157             throw new EJBException(ex);
158         }
159     }
160 
161     /*****************************************************************
162      * Business-Methods.
163      *****************************************************************/
164 
165     /**
166      * Processes the request.
167      * @param workerPK The primary key of the request.
168      * @throws FinderException If the request was not found.
169      * @throws CaException General CaException.
170      * @ejb.interface-method
171      *    view-type="remote"
172      */
173     public void processRequest(WorkerPK workerPK)
174         throws FinderException, CaException {
175 
176         RequestLocal request;
177         request = requestLocalHome.findByPrimaryKey(workerPK);
178         try {
179 
180             request.work(this);
181         } catch (EJBException ex) {
182             request.setFailed();
183             throw ex;
184         } catch (CaException ex) {
```

```
185             request.setFailed();
186             throw ex;
187         }
188 
189         try {
190 
191             switch (request.nextBench()) {
192             case Equipment.EXIT:
193                 request = null; // GC
194                 exitHome.create().processRequest(workerPK);
195                 break;
196             default:
197                 request.setFailed();
198             throw new FinderException
199             ("Unknown or incorrect next bench ("
200              + request.nextBench() + ").");
201             }
202         } catch (CreateException ex) {
203             request.setFailed();
204             throw new EJBException(ex);
205         } catch (RemoteException ex) {
206             request.setFailed();
207             throw new EJBException(ex);
208         }
209         request = null; // GC
210     }
211 
212     /**
213      * Reads an object from the card.
214      * @return The read object.
215      */
216     public Object readData() {
217 
218         Object obj = null;
219 
220         try {
221 
222             obj = pkcs11SessionHome.create().readData();
223         } catch (RemoteException ex) {
224             throw new EJBException(ex);
225         } catch (CreateException ex) {
226             throw new EJBException(ex);
227         }
228 
229         return obj;
230     }
231 
232     /**
233      * Write an object to the card.
234      * @param obj The object to be written to card.
235      */
236     public void writeData(Object obj) {
237 
238         try {
239 
240             pkcs11SessionHome.create().writeData(obj);
241         } catch (RemoteException ex) {
242             throw new EJBException(ex);
243         } catch (CreateException ex) {
244             throw new EJBException(ex);
245         }
246     }
247 
248     /**
249      * Do some calculations on an object.
250      * @param obj The object to be calculated.
251      * @return The modified object.
252      */
253     public Object calculate(Object obj) {
254 
255         try {
256 
```

```
257             obj = pkcs11SessionHome.create().calculate(obj);
258         } catch (RemoteException ex) {
259             throw new EJBException(ex);
260         } catch (CreateException ex) {
261             throw new EJBException(ex);
262         }
263
264         return obj;
265     }
266 }
```

Die Pkcs11Bench besitzt Referenzen auf die Home-Interfaces der `ExitEJB`, `RequestEJB` und die `Pkcs11SessionEJB`, die in der Methode `ejbCreate` (Zeile 127 ff.) über den `EJBServiceLocator` gesetzt und in der `ejbRemove` (Zeile 114 ff.) wieder freigegeben werden.

Die Businessmethode `processRequest` (Zeile 173 ff.) wird von den sich im Workflow vor der Pkcs11Bench befindlichen Benches aufgerufen und initiiert die Abarbeitung des nächsten Arbeitsschrittes des Requests. Der Ablauf untergliedert sich in folgende Schritte:

1. Erzeugung einer lokalen Instanz des zu bearbeitenden Requests in der eigenen Java-Virtual-Machine (Zeile 177 ff.).
2. Initiierung der Abarbeitung des nächsten Arbeitsschritts des Requests. Diesem wird eine Referenz auf die Instanz der Bench mitgegeben, wodurch er direkten Zugriff auf die von der Bench zur Verfügung gestellten Funktionen erhält (Zeile 180 ff.).
3. Nach dem Arbeitsschritt wird der Request auf die nächste von ihm spezifizierte Bench verschoben. Dieses Verschieben ist durch die Weitergabe des Primary-Keys des Requests realisiert (Zeile 191 ff.).

Weiteren Businessmethoden delegieren Aufrufe des Requests an den `Pkcs11Server`. Die Beanklasse ist eine Spezialisierung der Geschäftsklasse.

9.3.2 Die Klasse Pkcs11Bench

Die Klasse `domain.bench.Pkcs11Bench` stellt die Superklasse der `Pkcs11BenchEJB` dar und implementiert die von der Bench dem Request zur Verfügung zu stellenden Funktionalitäten.

Listing 9.2: domain.bench.Pkcs11Bench

```
1  package de.tud.cdc.flexiTrust.ca.domain.bench;
2
3  import jws.domain.bench.Bench;
4
5  /**
6   *
7   */
8  public abstract class Pkcs11Bench implements Bench {
9
10     /**
11      * Reads an object from the card.
12      * @return The read object.
13      */
14     public abstract Object readData();
15
16     /**
17      * Write an object to the card.
18      * @param obj The object to be written to card.
19      */
20     public abstract void writeData(Object obj);
21
22     /**
23      * Do some calculations on an object.
24      * @param obj The object to be calculated.
25      * @return The modified object.
26      */
27     public abstract Object calculate(Object obj);
28
29     /**
30      * A method usable by the worker.
31      */
32     public void doSomething() {
33
34         // Write some code here.
35     }
36 }
```

Beispiele für die dem Request zur Verfügung gestellten Funktionen sind in den Zeilen 14, 20, 27 und 32, bzw. aus der Klasse `domain.bench.CryptoBench` ersichtlich.

9.3.3 Die Klasse Equipment

Die Klasse `domain.bench.Equipment` beinhaltet eine Auflistung aller in der CA implementierten Benches. Die in ihr definierten Konstanten werden von den Requests und Workbenches zur Identifizierung der Benchtypen benötigt. Da wir hier eine neue Bench definieren, muß diese in der Klasse `Equipment` eingtragen werden.

Listing 9.3: domain.bench.Equipment

```
1  package de.tud.cdc.flexiTrust.ca.domain.bench;
2
3  public class Equipment {
4
5      /**
6       * Type: cryptobench
7       */
8      public static final int CRYPTOBENCH = 1;
9
10     /**
11      * Type: entrance
12      */
```

```
13      public static final int ENTRANCE = 2;
14
15      /**
16       * Type: exit
17       */
18      public static final int EXIT = 3;
19
20      /**
21       * Type: pkcs11bench
22       */
23      public static final int PKCS11BENCH = 4;
24  }
```

Die Zeile 23 definiert die Konstante der Pkcs11Bench.

9.3.4 Die Klasse EntranceEJB

Die Stateless-Session-Bean `ejbs.entrance.EntranceEJB` beinhaltet alle Operationen, die für die Bearbeitung von Issuer- und Admin-Requests in Verbindung mit weiteren Enterprise-JavaBeans notwendig sind. Sie stellt daher ein Bindeglied zwischen der in der Domänenklasse implementierten Logik und dem clusterweiten EJB-Tier der FlexiTRUST-CA dar. Für eine nähere Beschreibung dieser Klasse sei auf den Abschnitt 4.2.2 „Entrance“ verwiesen.

Für die in diesem Kapitel beschriebene Erweiterung der CA um eine neue Workbench muß diese der EntranceEJB als mögliches Ziel des Requests angegeben werden.

Listing 9.4: ejbs.entrance.EntranceEJB

```
1   package de.tud.cdc.flexiTrust.ca.ejbs.entrance;
2
3   [...] Other imports
4
5   import de.tud.cdc.flexiTrust.ca.interfaces.bench.Pkcs11BenchHome;
6
7   /**
8    * [...] Other XDoclet tags
9    *
10   * @ejb.ejb-ref
11   *     ejb-name="ejb/Pkcs11BenchEJB"
12   *
13   */
14  public class EntranceEJB
15      extends de.tud.cdc.flexiTrust.ca.domain.entrance.Entrance
16      implements SessionBean, BenchEJB {
17
18
19      [...] Other attributes
20
21      /**
22       * The Pkcs11BenchHome
23       */
24      private Pkcs11BenchHome pkcs11BenchHome;
25
26      /**
27       * ejbRemove
28       */
29      public void ejbRemove() {
30
31          [...]
32
```

```
33          pkcs11BenchHome = null;
34      }
35
36      /**
37       * ejbCreate
38       * @throws CreateException An entity EJB object could not be
39       * created.
40       */
41      public void ejbCreate()
42          throws CreateException {
43
44          [...] Other EJBServiceLocator lookups
45
46          try {
47
48              pkcs11BenchHome  =
49                  (Pkcs11BenchHome) EJBServiceLocator.lookupHome
50                  (Pkcs11BenchHome.class,
51                   "ejb/CDC.FlexiTRUST.Pkcs11BenchEJB");
52          } catch (NamingException ex) {
53              throw new EJBException(ex);
54          }
55      }
56
57      /*****************************************************************
58       * Business-Methods.
59       *****************************************************************/
60
61      [...] Other business methods
62
63      /**
64       * Processes the request.
65       * @param workerPK The primary key of the request.
66       * @throws CaException General CaException.
67       * @throws FinderException If the request was not found.
68       * @ejb.interface-method
69       *     view-type="remote"
70       */
71      public void processRequest(WorkerPK workerPK)
72          throws FinderException, CaException {
73
74          RequestLocal request;
75          request = requestLocalHome.findByPrimaryKey(workerPK);
76
77          [...]
78
79          try {
80
81              switch (request.nextBench()) {
82              case Equipment.CRYPTOBENCH:
83                  request = null; // GC
84                  cryptoBenchHome.create().processRequest(workerPK);
85                  break;
86              case Equipment.PKCS11BENCH:
87                  request = null; // GC
88                  pkcs11BenchHome.create().processRequest(workerPK);
89                  break;
90              case Equipment.EXIT:
91                  request = null; // GC
92                  exitHome.create().processRequest(workerPK);
93                  break;
94              default:
95                  request.setFailed();
96                  request = null; // GC
97                  throw new FinderException
98                      ("Unknown or incorrect next bench ("
99                       + request.nextBench() + ").");
100             }
101         } catch (CreateException ex) {
102             request.setFailed();
103             throw new EJBException(ex);
104         } catch (RemoteException ex) {
```

```
105             request.setFailed();
106             throw new EJBException(ex);
107         }
108         request = null; // GC
109     }
110
111     [...] Other business methods
112 }
```

Aus Performancegründen werden bereits bei der Instantiierung einer Enterprise-JavaBean alle Referenzen auf die Home-Interfaces benötigter Enterprise JavaBeans angelegt und in Attributen (Zeile 24) gespeichert. Da diese Referenzen für die Lebensdauer der Instanz lokal gehalten werden, müssen sie ensprechend verwaltet werden.

Für die neu hinzugekommene `Pkcs11BenchEJB` sind die XDoclet-Tags, die die Deployment-Deskriptoren beeinflussen, in den Zeilen 10 ff. dargestellt. Die Verwaltung der Referenz auf das Home-Interface innerhalb der Beanklasse wird in den Zeilen 46 ff. und 33 ff. implementiert. Es ist darauf zu achten, daß beim Entfernen einer Beaninstanz alle Referenzen auf andere Beans wieder freigegeben werden. Nachdem der Request auf der `EntranceEJB` abgearbeitet ist, wird er an die von ihm spezifizierte nächste Bench weitergegeben. Für den Fall, daß dieses die `Pkcs11Bench` ist, muß ein entsprechender Eintrag (Zeile 86 ff.) im Switch-Block erfolgen.

9.4 PKCS#11 Request

Beteiligte Komponenten:

Klasse	Beschreibung
`X509P11Request`	X509P11 Request
`X509P11RequestData`	Value-Object des `X509P11Requests`
`Request`	Geschäftsklasse der `RequestEJB`
`RequestEJB`	Allgemeine `RequestEJB`
`RequestDAO`	Data-Access-Object der `RequestEJB`
`Types`	Auflistung aller in der CA vorhandenen Requests

An der Enterprise-JavaBean `RequestEJB` müssen keine Modifikationen durchgeführt werden. Sie ist nur der Vollständigkeit halber in obiger Liste enthalten.

9.4.1 Die Klasse X509P11Request

Die Klasse `domain.request.X509P11Request` implementiert einen Pkcs#11-Request. Die grundlegende Struktur wird durch die Klasse `GenericWorker` definiert, welche die

Superklasse jedes Requests darstellt (Zeile 19). Hierdurch wird sichergestellt, daß ein Request folgende Methoden implementiert:

- `updateBeanData` und `getBeanRepresentation`: Aktualisieren bzw. Liefern der requestspezifischen Attribute. (Zeilen 62 ff. und 74 ff.)
- `loadResultSetFragment` und `storePreparedStatementFragment`: Behandlung der SQL-Statements (Zeilen 91 ff. und 107 ff.) der Klasse `RequestDAO`
- `initWork`: Die Initialisierung des Requests, sie wird von der EntranceEJB aufgerufen (Zeile 124 ff.)
- `work`: Die eigentliche Arbeitsmethode. Sie untergliedert sich innerhalb des Requests in die je nach Typ notwendigen Arbeitsschritte (Zeile 133 ff.)

Listing 9.5: domain.request.X509P11Request

```
1  package de.tud.cdc.flexiTrust.ca.domain.request;
2
3  import java.sql.PreparedStatement;
4  import java.sql.ResultSet;
5  import java.sql.SQLException;
6
7  import jws.domain.bench.Bench;
8  import jws.domain.worker.GenericWorker;
9  import jws.domain.worker.Worker;
10 import jws.domain.worker.data.WorkerData;
11
12 import de.tud.cdc.flexiTrust.ca.domain.bench.Equipment;
13 import de.tud.cdc.flexiTrust.ca.domain.request.data.X509P11RequestData;
14
15 /**
16  *
17  */
18 public class X509P11Request
19     extends GenericWorker {
20
21     /**
22      * Attribute
23      */
24     private int attribute;
25
26     /**
27      * The generic Request.
28      */
29     private Request request = null;
30
31     /**
32      * Constructor.
33      */
34     public X509P11Request() {
35
36     }
37
38     /**
39      * Constructor.
40      * @param request The generic Request
41      */
42     public X509P11Request(Request request) {
43
44         this.request = request;
45     }
```

```
46
47      /**
48       * Constructor.
49       * @param workerData The X509P11Request data.
50       * @param request The Request
51       */
52      public X509P11Request(WorkerData workerData, Request request) {
53
54          this.request = request;
55          updateBeanData(workerData);
56      }
57
58      /**
59       * Updates our bean data.
60       * @param workerData Bean data.
61       */
62      public void updateBeanData(WorkerData workerData) {
63
64          X509P11RequestData x509P11RequestData =
65              (X509P11RequestData) workerData;
66
67          attribute = x509P11RequestData.getAttribute();
68      }
69
70      /**
71       * Returns our bean representation.
72       * @return Our bean representation.
73       */
74      public WorkerData getBeanRepresentation() {
75
76          X509P11RequestData x509P11RequestData =
77              new X509P11RequestData();
78
79          x509P11RequestData.setAttribute(attribute);
80
81          return (WorkerData) x509P11RequestData;
82      }
83
84      /**
85       * Loads a part of the result set.
86       * @param rs Result set.
87       * @param offset Column offset.
88       * @return New column offset.
89       * @throws SQLException Database access error
90       */
91      public int loadResultSetFragment(ResultSet rs, int offset)
92          throws SQLException {
93
94          attribute = rs.getInt(offset++);
95
96          return offset;
97      }
98
99      /**
100      * Stores a part of the prepared statement.
101      * @param prepStmt Prepared statement
102      * @param offset Column offset.
103      * @return New column offset.
104      * @throws SQLException Database access error
105      */
106     public int storePreparedStatementFragment(PreparedStatement
107                                                   prepStmt,
108                                               int offset)
109         throws SQLException {
110
111         prepStmt.setInt(offset++, attribute);
112
113         return offset;
114     }
115
116     /*************************************************************
117      * Work-Methods.
```

```
118         ************************************************************/
119
120       /**
121        * Preprocessing. Perhaps for future use. Is called once bay the
122        * EntranceEJB
123        */
124       public void initWork() {
125
126           request.setNextBench(Equipment.PKCS11BENCH);
127       }
128
129       /**
130        * We decide which work has to be done now.
131        * @param bench The bench we reside on
132        */
133       public void work(Bench bench) {
134
135           switch (request.getState()) {
136           case Worker.DOSTEP1:
137               doStep1((Pkcs11Bench) bench);
138               break;
139           case Worker.DOSTEP2:
140               doStep2((Pkcs11Bench) bench);
141               break;
142           default:
143               request.setNextBench(Equipment.EXIT);
144               break;
145           }
146       }
147
148       /**
149        * Does the work of step one.
150        * @param bench The Pkcs11Bench
151        */
152       private void doStep1(Pkcs11Bench bench) {
153
154           obj = bench.readData();
155
156           request.setState(Worker.DOSTEP2);
157           request.setNextBench(Equipment.PKCS11BENCH);
158       }
159
160       /**
161        * Does the work of step two.
162        * @param bench The Pkcs11Bench
163        */
164       private void doStep2(Pkcs11Bench bench) {
165
166           obj = bench.calculate(obj);
167           bench.writeData(obj);
168
169           request.setState(Worker.DONE);
170           request.setNextBench(Equipment.EXIT);
171       }
172
173       /**
174        * Does the work of step one.
175        */
176       private void doStep1() {
177
178           // Your code here
179
180           request.setState(Worker.DOSTEP2);
181           request.setNextBench(Equipment.PKCS11BENCH);
182       }
183
184       /**
185        * Does the work of step two.
186        */
187       private void doStep2() {
188
189           // Your code here
```

```
190
191             request.setState(Worker.DONE);
192             request.setNextBench(Equipment.EXIT);
193         }
194     }
```

9.4.2 Die Klasse X509P11RequestData

Die Klasse `domain.request.data.X509P11RequestData` implementiert ein Value-Object und beinhaltet alle den speziellen Request bezeichnenden Attribute. Sie ist eine Spezialisierung der Klasse `jws.domain.worker.data.WorkerData`, in der die allgemeinen Attribute enthalten sind. Beide Klassen zusammengefaßt liefern eine Darstellung eines von der FlexiTRUST-CA zu bearbeitenden Requests, hier eines `X509P11-Requests`.

In der Klasse `WorkerData` sind folgende Attribute spezifiziert:

- `workerPK`: Der Primary-Key des Requests.
- `state`: Der aktuelle Status des Requests. Folgende Stati sind in der Klasse `jws.-domain.worker.Worker` definiert: `UNDEFINED`, `DOSTEP1..5`, `DONE` und `FAILED`.
- `creationDate`: Der Erzeugungszeitpunkt.
- `retirementDate`: Das Ablaufdatum.
- `type`: Typ des Requests (vgl. Abschnitt 9.4.5 „Die Klasse Types“).
- `sigDat`: Die signierten Daten.
- `nextBench`: Die aktuelle nächste Bench (vgl. Abschnitt 9.3.3 „Die Klasse Equipment“).

Weitere, für den Pkcs11-Request spezifische Attribute sind der `X509P11RequestData` hinzuzufügen, wobei für jedes der Attribute Accessormethoden implementiert werden müssen. Exemplarisch ist dieses im folgenden Listing dargestellt:

Listing 9.6: domain.request.data.X509P11RequestData

```
1  package de.tud.cdc.flexiTrust.ca.domain.request.data;
2
3  import java.io.Serializable;
4
5  import jws.domain.worker.data.WorkerData;
6
7  import de.tud.cdc.flexiTrust.ca.domain.bench.Equipment;
8  import de.tud.cdc.flexiTrust.ca.domain.request.Types;
9
10 public class X509P11RequestData
11     extends WorkerData
12     implements Serializable {
```

```
13
14      /**
15       * An attribute
16       */
17      private int attribute = 0;
18
19      /**
20       * Constructor.
21       */
22      public X509P11RequestData() {
23
24          super(Types.X509P11);
25      }
26
27      /**
28       * Get attribute.
29       * @return value of attribute.
30       */
31      public int getAttribute() {
32
33          return attribute;
34      }
35
36      /**
37       * Set the value of attribute.
38       * @param attribute Value to assign to attribute.
39       */
40      public void setAttribute(int attribute) {
41
42          this.attribute = attribute;
43      }
44  }
```

Wichtig neben der Attributdefinition und den Accessormethoden ist der Aufruf von `super` im Konstruktor. Der Parameter definiert den Typ des Requests (hier `X509P11`) .

9.4.3 Die Klasse Request

Die Klasse `domain.request.Request` beinhaltet allgemeine Funktionalitäten der Requests und muß an zwei Stellen geringfügig erweitert werden.

Listing 9.7: domain.request.Request

```
1   package de.tud.cdc.flexiTrust.ca.domain.request;
2
3   [...] Imports
4
5   public abstract class Request
6       extends Worker {
7
8       /**
9        * The specialized request.
10       */
11      private GenericWorker request = null;
12
13      /**
14       * Updates our bean data.
15       * @param workerData Bean data.
16       * @throws ASN1Exception An ASN.1 exception occured.
17       * @throws NoSuchAlgorithmException A requested particular
18       * cryptographic algorithm is not available.
19       * @throws IOException An I/O operation failed or was interrupted.
20       * @throws ClassNotFoundException No definition for the class to
21       * be loaded found.
22       * @throws CertificateEncodingException An error occured while
```

```
23          * encoding a certificate.
24          * @throws BadNameException An error occured during parsing
25          * of RFC1779 distinguished names.
26          */
27         public void updateBeanData(WorkerData workerData)
28             throws ASN1Exception, NoSuchAlgorithmException, IOException,
29                    ClassNotFoundException, CertificateEncodingException,
30                    BadNameException {
31 
32             super.updateBeanData(workerData);
33 
34             if (request == null) {
35 
36                 switch (super.getType()) {
37                 case Types.X509CRT:
38                     request = new X509CrtRequest(workerData, this);
39                     break;
40                 case Types.X509CRL:
41                     request = new X509CrlRequest(workerData, this);
42                     break;
43                 case Types.X509P11:
44                     request = new X509P11Request(workerData, this);
45                     break;
46                 case Types.ADMIN:
47                     request = new AdminRequest(workerData, this);
48                     break;
49                 case Types.ISSUER:
50                     request = new IssuerRequest(workerData, this);
51                     break;
52                 default:
53                     break;
54                 }
55             } else {
56                 request.updateBeanData(workerData);
57             }
58         }
59 
60         /**
61          * Loads a part of the result set.
62          * @param rs Result set.
63          * @param offset Column offset.
64          * @return New column offset.
65          * @throws SQLException Database access error
66          */
67         public int loadResultSetFragment(ResultSet rs, int offset)
68             throws SQLException {
69 
70             offset = super.loadResultSetFragment(rs, offset);
71 
72             if (request == null) {
73                 switch (super.getType()) {
74                 case Types.X509CRT:
75                     request = new X509CrtRequest(this);
76                     break;
77                 case Types.X509CRL:
78                     request = new X509CrlRequest(this);
79                     break;
80                 case Types.X509P11:
81                     request = new X509P11Request(this);
82                     break;
83                 case Types.ADMIN:
84                     request = new AdminRequest(this);
85                     break;
86                 case Types.ISSUER:
87                     request = new IssuerRequest(this);
88                     break;
89                 default:
90                     break;
91                 }
92             }
93             offset = request.loadResultSetFragment(rs, offset);
94 
```

```
95          return offset;
96      }
97
98      [...]
99  }
```

Die Methoden `updateBeanData` und `loadResultSetFragment` (Zeilen 43 ff. und 80 ff.) müssen jeweils um den Fall `Types.X509P11` erweitert werden. Dies ist notwendig, da beide Methoden zum Zeitpunkt der Instantiierung der `RequestEJB` aufgerufen werden. In diesem Moment existiert noch keine Instanz des `X509P11Requests`, die hier erzeugt wird.

9.4.4 Die Klasse RequestDAO

Die Klasse `ejb.request.RequestDAO` implementiert ein Data-Access-Object für den Datenbankzugriff der `RequestEJB`. Innerhalb des DAOs werden vier Fälle unterschieden:

- Insert: Einfügen eines Datensatzes in die Datenbank,
- Select: Lesen eines Datensatzes,
- Update: Aktualisieren eines Datensatzes,
- Delete: Löschen eines Datensatzes.

Mindestens die ersten drei Fälle müssen für neue Requests implementiert werden.

Listing 9.8: ejbs.request.RequestDAO

```
1   package de.tud.cdc.flexiTrust.ca.ejbs.request;
2
3   [...] Imports
4
5   public class RequestDAO
6       extends DataAccessObject
7       implements WorkerDAO {
8
9       /**
10       * This constant describes the SQL error code for a duplicate
             primary key.
11       */
12      private static final int ERR_DUPLICATE_KEY = 1062;
13
14      /**
15       * Constructor.
16       * @throws NamingException If the lookup failed.
17       */
18      public RequestDAO()
19          throws NamingException {
20
21          super();
22      }
23
24      /*************************************************************
25       * Finder-Methods.
```

```
26          ************************************************************/
27
28      [....]
29
30      /************************************************************
31       * Request-Methods.
32       ************************************************************/
33
34      /**
35       * Removes a Request.
36       * @param con Connection.
37       * @param workerPK The primary key of the request.
38       * @param worker The Request.
39       * @throws SQLException Database access error
40       */
41      public void deleteWorker(Connection con,
42                               WorkerPK workerPK,
43                               Worker worker)
44          throws SQLException {
45
46          String deleteStatement =
47              "delete from requests where requestPK = ? ";
48
49          PreparedStatement prepStmt =
50              con.prepareStatement(deleteStatement);
51
52          workerPK.storePreparedStatementFragment(prepStmt, 1);
53
54          int rowCount = prepStmt.executeUpdate();
55
56          prepStmt.close();
57
58          if (rowCount == 0) {
59              throw new NoSuchEntityException
60                  ("Remove for request " + workerPK + " failed.");
61          }
62
63          switch (worker.getType()) {
64          case Types.X509CRT:
65              deleteX509CrtRequest(con, workerPK);
66              break;
67          case Types.X509CRL:
68              deleteX509CrlRequest(con, workerPK);
69              break;
70          case Types.X509P11:
71              deleteX509P11Request(con, workerPK);
72              break;
73          default:
74              break;
75          }
76      }
77
78      /**
79       * Inserts a Request.
80       * @param con Connection.
81       * @param workerPK The primary key of the Request.
82       * @param worker The Request.
83       * @throws SQLException Database access error
84       * @throws DuplicateKeyException An entity EJB object with the same
85       * key already exists.
86       */
87      public void insertWorker(Connection con,
88                               WorkerPK workerPK,
89                               Worker worker)
90          throws SQLException, DuplicateKeyException {
91
92          String insertStatement =
93              "insert into requests ("
94              + "requestPK, state, creationDate, retirementDate, type, "
95              + "sigDat, nextBench) "
96              + "values (?, ?, ?, ?, ?, ?, ?)";
97
```

```
98          PreparedStatement prepStmt =
99              con.prepareStatement(insertStatement);
100
101         int offset = workerPK
102             .storePreparedStatementFragment(prepStmt, 1);
103
104         worker.storePreparedStatementFragment(prepStmt, offset);
105
106         int rowCount = 0;
107
108         try {
109             rowCount = prepStmt.executeUpdate();
110         } catch (SQLException ex) {
111             prepStmt.close();
112             if (ex.getErrorCode() == ERR_DUPLICATE_KEY) {
113                 throw new DuplicateKeyException
114                     ("The request with the primary key "
115                      + workerPK + " allready exists.");
116             } else {
117                 throw ex;
118             }
119         }
120
121         prepStmt.close();
122
123         if (rowCount == 0) {
124             throw new NoSuchEntityException
125                 ("Insert for request " + workerPK + " failed.");
126         }
127
128         switch (worker.getType()) {
129         case Types.X509CRT:
130             insertX509CrtRequest(con, workerPK, worker);
131             break;
132         case Types.X509CRL:
133             insertX509CrlRequest(con, workerPK, worker);
134             break;
135         case Types.X509P11:
136             insertX509P11Request(con, workerPK, worker);
137             break;
138         case Types.ADMIN:
139             insertAdminRequest(con, workerPK, worker);
140             break;
141         case Types.ISSUER:
142             insertIssuerRequest(con, workerPK, worker);
143             break;
144         default:
145             break;
146         }
147     }
148
149     /**
150      * Loads a Request.
151      * @param con Connection.
152      * @param workerPK The primary key of the Request.
153      * @param worker The Request.
154      * @throws SQLException Database access error
155      */
156     public void selectWorker(Connection con,
157                              WorkerPK workerPK,
158                              Worker worker)
159         throws SQLException {
160
161         String selectStatement = null;
162
163         switch (workerPK.getType()) {
164         case Types.X509CRT:
165             selectStatement = "select "
166                 + "state, creationDate, retirementDate, type, sigDat, "
167                 + "nextBench, "
168                 + "x509Certificate, revocPass, privKey, keyPass "
169                 + "from requests R, x509CrtRequests X "
```

```
170                 + "where R.requestPK = ? "
171                 + "and R.requestPK = X.requestPK ";
172             break;
173         case Types.X509CRL:
174             selectStatement = "select "
175                 + "state, creationDate, retirementDate, type, sigDat, "
176                 + "nextBench, "
177                 + "issuer, crlEntry, revocPass, x509Crl "
178                 + "from requests R, x509CrlRequests X "
179                 + "where R.requestPK = ? "
180                 + "and R.requestPK = X.requestPK ";
181             break;
182         case Types.X509P11:
183             selectStatement = "select "
184                 + "state, creationDate, retirementDate, type, sigDat, "
185                 + "nextBench, "
186                 + "attribute "
187                 + "from requests R, x509P11Requests X "
188                 + "where R.requestPK = ? "
189                 + "and R.requestPK = X.requestPK ";
190             break;
191         case Types.ADMIN:
192             selectStatement = "select "
193                 + "state, creationDate, retirementDate, type, sigDat, "
194                 + "nextBench, "
195                 + "adminDN, adminPrivKey, certificate, certSerial, "
196                 + "issuerDN, validFrom, validUntil, revocPass, "
197                 + "keyPass, sigAlg, secProv, certChain "
198                 + "from requests R, adminRequests A "
199                 + "where R.requestPK = ? "
200                 + "and R.requestPK = A.requestPK";
201             break;
202         case Types.ISSUER:
203             selectStatement = "select "
204                 + "state, creationDate, retirementDate, type, sigDat, "
205                 + "nextBench, "
206                 + "issuerDN, issuerPrivKey, certificate, certSerial, "
207                 + "revocPass, keyPass, "
208                 + "sigAlg, secProv, certChain "
209                 + "from requests R, issuerRequests I "
210                 + "where R.requestPK = ? "
211                 + "and R.requestPK = I.requestPK";
212             break;
213         default:
214             throw new SQLException("Unkown request type.");
215         }
216 
217         PreparedStatement prepStmt =
218             con.prepareStatement(selectStatement);
219 
220         workerPK.storePreparedStatementFragment(prepStmt, 1);
221 
222         ResultSet rs = prepStmt.executeQuery();
223 
224         if (rs.next()) {
225             worker.loadResultSetFragment(rs, 1);
226             prepStmt.close();
227         } else {
228             prepStmt.close();
229             throw new NoSuchEntityException
230                 ("Row for request " + workerPK + " not found.");
231         }
232     }
233 
234     /**
235      * Updates a Request.
236      * @param con Connection.
237      * @param workerPK The primary key of the Request.
238      * @param worker The Request.
239      * @throws SQLException Database access error
240      */
241     public void updateWorker(Connection con,
```

```
242                             WorkerPK workerPK,
243                             Worker worker)
244         throws SQLException {
245
246         String updateStatement = "update requests "
247             + "set state = ?, creationDate = ?, retirementDate = ?, "
248             + "type = ?, sigDat = ?, nextBench = ? "
249             + "where requestPK = ?";
250
251         PreparedStatement prepStmt =
252             con.prepareStatement(updateStatement);
253
254         int offset =
255             worker.storePreparedStatementFragment(prepStmt, 1);
256
257         workerPK.storePreparedStatementFragment(prepStmt, offset);
258
259         int rowCount = prepStmt.executeUpdate();
260
261         prepStmt.close();
262
263         if (rowCount == 0) {
264             throw new NoSuchEntityException
265                 ("Update for request " + workerPK + " failed.");
266         }
267
268         switch (worker.getType()) {
269         case Types.X509CRT:
270             updateX509CrtRequest(con, workerPK, worker);
271             break;
272         case Types.X509CRL:
273             updateX509CrlRequest(con, workerPK, worker);
274             break;
275         case Types.X509P11:
276             updateX509P11Request(con, workerPK, worker);
277             break;
278         case Types.ADMIN:
279             updateAdminRequest(con, workerPK, worker);
280             break;
281         case Types.ISSUER:
282             updateIssuerRequest(con, workerPK, worker);
283             break;
284         default:
285             break;
286         }
287     }
288
289     /**
290      * Sets the state to failed.
291      * @param con Connection
292      * @param workerPK The primary key of the request.
293      * @throws SQLException Database access error
294      */
295     public void setFailed(Connection con,
296                           WorkerPK workerPK)
297         throws SQLException {
298
299         String updateStatement = "update requests "
300             + "set state = ? "
301             + "where requestPK = ?";
302
303         PreparedStatement prepStmt =
304             con.prepareStatement(updateStatement);
305
306         int offset = 1;
307         prepStmt.setInt(offset++, Worker.FAILED);
308         workerPK.storePreparedStatementFragment(prepStmt, offset);
309
310         int rowCount = prepStmt.executeUpdate();
311
312         prepStmt.close();
313
```

```
314         if (rowCount == 0) {
315             throw new NoSuchEntityException
316                 ("Update for request " + workerPK + " failed.");
317         }
318     }
319 
320     /*************************************************************
321      * X509CrtRequest-Methods.
322      *************************************************************/
323 
324     [...]
325 
326     /*************************************************************
327      * X509CrlRequest-Methods.
328      *************************************************************/
329 
330     [...]
331 
332     /*************************************************************
333      * X509P11Request-Methods.
334      *************************************************************/
335 
336     /**
337      * Removes a Request.
338      * @param con Connection.
339      * @param workerPK The primary key of the request.
340      * @throws SQLException Database access error
341      */
342     private void deleteX509P11Request(Connection con,
343                                       WorkerPK workerPK)
344         throws SQLException {
345 
346         String deleteStatement =
347             "delete from x509P11Requests where requestPK = ? ";
348 
349         PreparedStatement prepStmt =
350             con.prepareStatement(deleteStatement);
351 
352         workerPK.storePreparedStatementFragment(prepStmt, 1);
353 
354         int rowCount = prepStmt.executeUpdate();
355 
356         prepStmt.close();
357 
358         if (rowCount == 0) {
359             throw new NoSuchEntityException
360                 ("Remove for request " + workerPK + " failed.");
361         }
362     }
363 
364 
365     /**
366      * Inserts a Request.
367      * @param con Connection.
368      * @param workerPK The primary key of the Request.
369      * @param worker The Request.
370      * @throws SQLException Database access error
371      */
372     private void insertX509P11Request(Connection con,
373                                       WorkerPK workerPK,
374                                       Worker worker)
375         throws SQLException {
376 
377         String insertStatement =
378             "insert into x509P11Requests ("
379             + "requestPK, attribute) "
380             + "values (?, ?)";
381 
382         PreparedStatement prepStmt =
383             con.prepareStatement(insertStatement);
384 
385         int offset = workerPK
```

```
386             .storePreparedStatementFragment(prepStmt, 1);
387
388         worker.storePreparedConcreteStatementFragment(prepStmt,offset);
389
390         int rowCount = prepStmt.executeUpdate();
391
392         prepStmt.close();
393
394         if (rowCount == 0) {
395             throw new NoSuchEntityException
396                 ("Insert for request " + workerPK + " failed.");
397         }
398     }
399
400     /**
401      * Updates a Request.
402      * @param con Connection.
403      * @param workerPK The primary key of the Request.
404      * @param worker The Request.
405      * @throws SQLException Database access error
406      */
407     private void updateX509P11Request(Connection con,
408                                       WorkerPK workerPK,
409                                       Worker worker)
410         throws SQLException {
411
412         String updateStatement = "update x509P11Requests "
413             + "set attribute = ? "
414             + "where requestPK = ?";
415
416         PreparedStatement prepStmt =
417             con.prepareStatement(updateStatement);
418
419         int offset =
420             worker.storePreparedConcreteStatementFragment(prepStmt, 1);
421
422         workerPK.storePreparedStatementFragment(prepStmt, offset);
423
424         int rowCount = prepStmt.executeUpdate();
425
426         prepStmt.close();
427
428         if (rowCount == 0) {
429             throw new NoSuchEntityException
430                 ("Update for request " + workerPK + " failed.");
431         }
432     }
433
434     /*****************************************************************
435      * IssuerRequest-Methods.
436      *****************************************************************/
437
438     [...]
439
440     /*****************************************************************
441      * AdminRequest-Methods.
442      *****************************************************************/
443
444     [...]
445 }
```

Die Methoden in den Zeilen 41, 87, 156 und 241 werden von der Beanimplementierung aufgerufen. Sie führen den eigentlichen Datenbankzugriff aus. Für die requestspezifischen Daten wird an spezielle Methoden delegiert (70, 135 und 275). Eine Implementierung der Pkcs11-spezifischen Methoden ist in den Zeilen 182 ff. und 342 ff. dargestellt.

9.4.5 Die Klasse Types

Die Klasse `domain.request.Types` beinhaltet eine Auflistung aller in der CA implementierten Requests. Die in ihr definierten Konstanten werden von den Requests und Workbenches zur Identifizierung verwendet. Da wir einen neuen Request definieren muß dieser hier eingetragen werden.

Listing 9.9: domain.request.Types

```
1  package de.tud.cdc.flexiTrust.ca.domain.request;
2
3  public class Types {
4
5      /**
6       * Type: admin request
7       */
8      public static final int ADMIN = 1;
9
10     /**
11      * Type: issuer request
12      */
13     public static final int ISSUER = 2;
14
15     /**
16      * Type: x509 crt request
17      */
18     public static final int X509CRT = 3;
19
20     /**
21      * Type: x509 crl request
22      */
23     public static final int X509CRL = 4;
24
25     /**
26      * Type: x509 p11 request
27      */
28     public static final int X509P11 = 5;
29 }
```

Die Zeile 28 definiert die Konstante des X509P11Requests.

9.5 PKCS#11 MBean-Service

Beteiligte Komponenten:

Klasse	Beschreibung
`Pkcs11CardAccess`	MBean-Klasse des `CardAccess`.
`Pkcs11SessionEJB`	Beanklasse `Pkcs11SessionEJB`.

Die nachfolgenden Abschnitte beschreiben die beteiligten Komponenten zur Realisierung des Hardwarezugriffs von Pkcs11 und ihre Aufgaben entsprechend ihrer Zuordnung zu den verschiedenen Schichten der J2EE-Architektur.

9.5.1 Die Klasse Pkcs11CardAccess

Die Klasse `pkcs11server.cardaccess.Pkcs11CardAccess` soll einen Service für den Zugriff auf die SmartCard-Hardware zur Verfügung stellen. Sie wird analog zu den Services des Pkcs7Service als MBean implementiert.

Listing 9.10: pkcs11server.cardaccess.Pkcs11CardAccess

```
1  package de.tud.cdc.flexiTrust.ca.pkcs11server.cardaccess;
2
3  import javax.naming.InitialContext;
4  import javax.naming.NamingException;
5  import javax.naming.NameNotFoundException;
6  import javax.naming.Reference;
7  import javax.naming.StringRefAddr;
8
9  import org.jboss.naming.NonSerializableFactory;
10 import org.jboss.system.ServiceMBeanSupport;
11
12 /**
13  * @jmx.mbean
14  *    description="Pkcs11CardAccess Service"
15  *    name=":service=Pkcs7ExitService"
16  *    extends="org.jboss.system.ServiceMBean"
17  */
18 public class Pkcs11CardAccess
19     extends ServiceMBeanSupport
20     implements Pkcs11CardAccessMBean {
21
22     /**
23      * The JndiName for this MBean
24      */
25     public static final String JNDI_NAME =
26         "java:/Pkcs11CardAccessService";
27
28     /**
29      * A property
30      */
31     private long property = -1;
32
33     /**
34      * Starts this MBean service
35      * @throws NamingException If the lookup failed.
36      */
37     public void startService()
38         throws NamingException {
39
40         // Add security provider
41
42         // Get properties from System-Properties-Service
43
44         // Instantiate a thread if needed and start it.
45
46         InitialContext ctx = new InitialContext();
47
48         //bind myself into JNDI, at java:/Pkcs11CardAccessService
49         NonSerializableFactory.bind(JNDI_NAME, this);
50         StringRefAddr addr = new StringRefAddr("nns", JNDI_NAME);
51         Reference ref =
52             new Reference(this.getClass().getName(),
53                           addr,
54                           NonSerializableFactory.class.getName(),
55                           null);
56         ctx.bind(JNDI_NAME, ref);
57     }
58
59     /**
60      * Stops this MBean service
61      * @throws NamingException If the lookup failed.
```

```
62          * @throws NameNotFoundException If the jndi name was not found.
63          */
64         public void stopService()
65             throws NamingException, NameNotFoundException {
66
67             InitialContext ctx = new InitialContext();
68
69             ctx.unbind(JNDI_NAME);
70             NonSerializableFactory.unbind(JNDI_NAME);
71
72             // Stop the thread if it was created and set it to null
73         }
74
75         /**************************************************************
76          * Business-Methods.
77          **************************************************************/
78
79         /**
80          * Reads an object from the card.
81          * @return The read object.
82          */
83         public Object readData() {
84
85             Object obj = null;
86
87             // Read an object from the card.
88
89             return obj;
90         }
91
92         /**
93          * Write an object to the card.
94          * @param obj The object to be written to card.
95          */
96         public void writeData(Object obj) {
97
98             // Write an object on the card.
99         }
100
101        /**
102         * Do some calculations on an object.
103         * @param obj The object to be calculated.
104         * @return The modified object.
105         * @ejb.interface-method
106         *    view-type="remote"
107         */
108        public Object calculate(Object obj) {
109
110            // Do some calculations on the card.
111
112            return obj;
113        }
114
115        /**************************************************************
116         * Set/Get-Methods.
117         **************************************************************/
118
119        /**
120         * Set a property
121         * @param property The property
122         * @jmx.managed-attribute
123         */
124        public void setProperty(long property) {
125
126            this.property = property;
127        }
128
129        /**
130         * Get property
131         * @return The property
132         * @jmx.managed-attribute
133         */
```

```
134     public long getProperty() {
135
136         return property;
137     }
138 }
```

In den Zeilen 13 und ff. werden, analog zu den bereits besprochenen XDoclet-Tags der Enterprise-JavaBeans, entsprechende JMX-Tags definiert. Die Methode `startService` (Zeile 37 ff.), die vom Application-Server beim Systemstart aufgerufen wird, ist für die initiale Konfiguration des Services, das Starten eventuell benötigter Threads und das Einbinden in den JNDI-Namespace verantwortlich. Das Gegenstück dazu stellt die Methode `stopService` (Zeile 64 ff.) dar.

Für Attribute, die zur Laufzeit des Services über die JMX-Schnittstelle konfigurierbar sein sollen, sind `get/set`-Methoden dem Beispiel im Listing entsprechend (Zeilen 124 ff.) vorzusehen.

9.5.2 Die Klasse Pkcs11SessionEJB

Die Klasse `pkcs11server.session.Pkcs11Session` stellt analog zur Klasse `Pkcs7-SessionEJB` die Schnittstelle zwischen dem Service und dem EJB-Tier der Flexi-TRUST-CA dar.

Listing 9.11: pkcs11server.session.Pkcs11SessionEJB

```
1  package de.tud.cdc.flexiTrust.ca.pkcs11server.session;
2
3  import javax.ejb.CreateException;
4  import javax.ejb.EJBException;
5  import javax.ejb.SessionBean;
6  import javax.ejb.SessionContext;
7  import javax.naming.NamingException;
8  import javax.naming.InitialContext;
9
10 import de.tud.cdc.flexiTrust.ca.pkcs11server.cardaccess.
       Pkcs11CardAccess;
11
12 /**
13  * @ejb.bean
14  *     display-name="Pkcs11SessionEJB"
15  *     name="ejb/Pkcs11SessionEJB"
16  *     jndi-name="ejb/CDC.FlexiTRUST.Pkcs11SessionEJB"
17  *     type="Stateless"
18  *     transaction-type="Container"
19  *     reentrant="false"
20  *     use-soft-locking="true"
21  *     view-type="remote"
22  *
23  * @ejb.home
24  *     remote-class="de.tud.cdc.flexiTrust.ca.pkcs11server.session.
        Pkcs11SessionHome"
25  *
26  * @ejb.interface
27  *     remote-class="de.tud.cdc.flexiTrust.ca.pkcs11server.session.
        Pkcs11Session"
28  *
29  * @ejb.transaction
30  *     type="Required"
```

```
 *
 * @jboss.clustered
 *
 * @jboss.cluster-config
 *    partition-name="_PARTITION_"
 *    home-policy="org.jboss.ha.framework.interfaces.RandomRobin"
 *    bean-policy="org.jboss.ha.framework.interfaces.RandomRobin"
 */
public class Pkcs11SessionEJB
    implements SessionBean {

    /**
     * The SessionContext.
     */
    private SessionContext ctx;

    /**
     * The Pkcs11CardAccessService
     */
    private Pkcs11CardAccess pkcs11CardAccess = null;

    /**
     * Set the SessionContext. The Container calls this method after
     * creation of the instance.
     * @param context The SessionContext for this instance.
     */
    public void setSessionContext(SessionContext context) {

        ctx = context;
        pkcs11CardAccess = null;
    }

    /**
     * ejbActivate.
     */
    public void ejbActivate() {

    }

    /**
     * ejbPassivate
     */
    public void ejbPassivate() {

    }

    /**
     * ejbRemove
     */
    public void ejbRemove() {

        ctx = null;
    }

    /**
     * ejbCreate
     * @throws CreateException An entity EJB object could not be
     * created.
     */
    public void ejbCreate()
        throws CreateException {

        try {

            InitialContext ic = new InitialContext();

            pkcs11CardAccess =
                (Pkcs11CardAccess) ic.lookup
                ("java:/Pkcs11CardAccessService");
        } catch (NamingException ex) {
            throw new EJBException(ex);
        }
```

```
103     }
104
105     /*****************************************************************
106      * Business-Methods.
107      *****************************************************************/
108
109     /**
110      * Reads an object from the card.
111      * @return The read object.
112      * @ejb.interface-method
113      *     view-type="remote"
114      */
115     public Object readData() {
116
117         Object obj = null;
118
119         obj = pkcs11CardAccess.readData();
120
121         return obj;
122     }
123
124     /**
125      * Write an object to the card.
126      * @param obj The object to be written to card.
127      * @ejb.interface-method
128      *     view-type="remote"
129      */
130     public void writeData(Object obj) {
131
132         pkcs11CardAccess.writeData(obj);
133     }
134
135     /**
136      * Do some calculations on an object.
137      * @param obj The object to be calculated.
138      * @return The modified object.
139      * @ejb.interface-method
140      *     view-type="remote"
141      */
142     public Object calculate(Object obj) {
143
144         obj = pkcs11CardAccess.calculate(obj);
145
146         return obj;
147     }
148 }
```

9.6 Buildfiles

Für die in den letzen Abschnitten beschriebenen Erweiterungen der FlexiTRUST-CA müssen neue Buildfiles für Ant geschrieben bzw. existierende angepaßt werden.

Neue Buildfiles werden für die Packages `pkcs11server.cardaccess` und `pkcs11-server.session` benötigt. Als Vorlage hierzu können die entsprechenden Buildfiles des `pkcs7server` verwendet werden. Weiterhin muß das zentrale Buildfile in `$FLEXI-TRUST_HOME/ca/src` geringfügig geändert werden. Diese Änderungen sind analog zu den Targets des `Pkcs7Servers` zu implementieren.

Für die durchgeführten Erweiterungen des EJB-Tiers muß and den Buildfiles nichts geändert werden.

Kapitel 10

Abschließende Bemerkungen

MW+LF

10.1 Ausblick

Während der Spezifizierung und Implementierung der FlexiTRUST-CA und allen damit zusammenhängenden Aufgaben sind Probleme aufgetreten, die bisher nicht bzw. nicht adäquat gelöst werden konnten. Diese Probleme und mögliche Lösungsansätze werden im folgenden Abschnitt ebenso beschrieben wie weitere Ideen, die während der Implementierungsphase entstanden aber nicht mehr umgesetzt werden konnten.

10.1.1 Datenbank

MySQL unterstützt in der verwendeten Version keine „nested queries“, also Queries, die anstatt einer oder mehrerer Tabellen eine weitere Query zur Auswertung verwenden. Diese Funtion wird zur Zeit implementiert und wird voraussichtlich ab Version 4.1 verfügbar sein. Das davon betroffene zukünftige Selectstatement in der Methode `getValidCertificates` der Klasse `CaStockDAO` ist bereits auskommentiert eingefügt.

Weiterhin ist die Verwendung von Replikanten eine Art Behelf, solange noch keine hinreichend bewährten und günstigen clusterfähigen verfügbar sind.

Um bei Ausfall des Masters den Replikanten unmittelbar als zu verwendenden Datenbank-Server anschliessen zu können, muß der Replikant dafür konfiguriert werden. Zur Zeit ist der Replikant nur für Replizierungsaufgaben konfiguriert, da nicht genügend Zeit blieb, adäquate Tests durchzuführen. Grundsätzlich sollte es hier ausreichen, Binary-Logging zu aktivieren und den instandgesetzten bisherigen Master als neuen Replikanten über Änderungen an dessen Konfigurationsdatei an den neuen Master anzuschliessen.

Momentan ist es leider noch nicht möglich, einen aktuellen Snapshot des Masters zu verwenden, da der Replikant beim Start versuchen würde, die bereits erstellten Tabellen erneut zu erstellen. Da beim Start mit der initialen Datenbank sämtliche Daten über das Binary-Log aktualisiert werden, ist dies nicht weiter von Bedeutung, eine andere Lösung wäre jedoch wünschenswert.

Eine andere Möglichkeit stellt Two-Way-Replication dar. Das bedeutet, den Replikanten so zu konfigurieren, daß er Binary-Logs erstellt, die jede Änderung, außer den vom Master erhaltenen Änderungen über dessen Binary-Log, enthalten. Ebenso würde der Master so konfiguriert, sich über die Binary-Logs des Replikanten zu aktualisieren. Dadurch wäre es möglich, bei Ausfall des Masters durch Änderung der Datei `$JBOSS_HOME/server/flexiTRUST/deploy/mysql-service.xml` JBoss auf den Replikanten einzustellen. Durch das Autodeployment sollte JBoss dann auf den Replikanten eingestellt sein. Ob dies tatsächlich bei laufendem JBoss funktioniert, müsste allerdings getestet werden. Auf jeden Fall muß verhindert werden, daß vor korrektem Anschluss des Replikanten weitere Requests ins System kommen. Der bisherige Master könnte dann als Replikant an den neuen Master angeschlossen werden, da der neue Master bereits so konfiguriert ist, Änderungen in ein Binary-Log zu schreiben über das sich der neue Replikant aktualisieren kann. Der Grund, warum die replizierten Änderungen vom Replikanten nicht geloggt werden dürfen ist, daß der alte Master diese Änderungen bereits durchgeführt hat und somit nicht erneut durch Replikation durchführen darf.

Desweiteren sind zum Thema Datenbankgröße Überlegungen anzustellen. Da die Tabellen certificates, privKeys und sämtliche Requesttabellen regelmäßig größer werden und nie Daten aus ihnen gelöscht werden, muß eine Lösung gefunden werden, wie der Datenbestand dennoch regelmäßig reduziert werden kann, ohne daß der Betrieb der CA gefährdet ist. Eventuell ist es tragbar, alle Daten, die zu abgelaufenen Zertifikaten bestimmten Alters gehören, derart zu exportieren, daß sie mit vertretbarem Aufwand wiederhergestellt und damit für die CA zur Verfügung gestellt werden können.

10.1.2 Workbenches und Requests als Plug-In

Trotz der Modularität der im Rahmen dieser Arbeit umgesetzten FlexiTRUST-CA darf bei einer Erweiterung der Überblick über das Gesamtsystem nicht verloren gehen. Eine Art Plug-In-Architetur, die es ermöglicht, eine weitere Workbench oder einen weiteren Request durch einfaches Hineinkopieren zu aktivieren, wäre durchaus wünschenswert. Hierdurch würde es auch einfach möglich, unterschiedliche Lizensierungsmodelle umzusetzen. Je nach Lizenz werden Plug-Ins hinzugegeben, entfernt oder durch modifzierte Versionen ersetzt.

Ein erster Ansatz wäre, eine Art zentrale Factory zu implementieren, bei der sich die Plug-Ins registrieren und somit den anderen Komponenten zur Verfügung stellen. Als Basis kann hierfür der bereits umgesetzte `EJBServiceLoctor` dienen.

Eine weitere Möglichkeit könnte in der Implementierung eines MBean-Services bestehen, der dem Gesamtsystem eine Objektstruktur mit Informationen über installierte Plug-Ins zur Verfügung stellt. Sofern diese Struktur serialisierbar implementiert wird, kann sie an den globalen HA-JNDI gebunden werden und stünde somit auch clusterweit zur Verfügung.

10.1.3 CaStock

Momentan werden sämtliche Administratoren- und Issuer-Requests nicht auf der Cryptobench verifiziert. Grund hierfür ist, daß bei der Einbringung von initialem Issuer/Administrator keine Daten zur Verifikation oder Signierung im Stock vorliegen. Die initiale Einbringung geht aber genauso vonstatten wie spätere Issuer/Administratoren eingebracht werden. Daher wurde zunächst auf eine Verifikation dieser Requests verzichtet, Ansätze zur Änderung wären ein zusätzlicher `InitKeyRequest` mit einer zusätzlichen `InitKeyRequestData` im EJB-Tier, hierfür müsste allerdings eine Tabelle für einen einzigen Datensatz angelegt werden. Da dies nicht akzeptabel ist, wurde auf diesen Ansatz bisher verzichtet, möglicherweise kann er dennoch unter Umgehung einer zusätzlichen Tabelle verfolgt werden.

Issuer und Administrator nehmen nicht aktiv an der Requestverarbeitung teil, daher wurden sie in das Package `stock` integriert. Diese Lösung ist aber eher als eine Behelfslösung anzusehen. Da Issuer und Administrator eine Art Personal der CA darstellen, könnten sie möglicherweise in ein Package `staff` verschoben werden.

In der weiteren Entwicklung könnte noch im Zertifikatsantrag von der RA definiert werden, ob der private Schlüssel des Users gespeichert werden soll.

10.1.4 Benutzerschnittstelle

In einer eventuellen späteren Erweiterung der FlexiTRUST-CA um ein Administrations- oder UserInterface wird eine Web-Architektur mit JavaServlet und JavaServerPages basierend auf dem Model-View-Controller (MVC) Design-Paradigma vorgeschlagen. Diese Technologie wird u.a. von dem Struts-Framework unterstützt. Nähere Informationen zu Struts und MVC können [Fou02], [EG01] und [Oes01] entnommen werden.

10.2 Fazit

Mit dieser Diplomarbeit ist eine leistungsfähige und moderne Certification Authority für FlexiTRUST entstanden, die aufgrund ihrer Modularität und ihrer auf JWorkShop basierenden Architektur leicht erweiterbar ist.

Der neu implementierte JWorkShop liefert eine Architektur, in der auch andere Bereiche von FlexiTRUST neu entwickelt werden können, um eine Heterogenität bezüglich Architektur und Technologien bei den Hauptkomponenten von FlexiTRUST zu erreichen.

Abkürzungsverzeichnis

API Application Programming Interface

CA Certification Authority

CRL Certificate Revocation List

CRLEntry Certificate Revocation List Entry

CVS Concurrent Versions System

DAO Data Access Object

DV Datenverarbeitung

EAS Enterprise Application Server

EIS Enterprise Information Systems

EJB Enterprise Java Bean

HA-JNDI High Availability Java Namnig and Directory Interface

HTML HyperText Markup Language

IP Internet Protocol

IS Infrastructure Services

J2EE Java 2 Platform Enterprise Edition

J2SE Java 2 Platform Standard Edition

JAAS Java Authentication and Authorization Service

JDE Java Development Environment

JMX Java Management Extensions

JNDI Java Naming and Directory Interface

JSP Java Server Pages

JTA Java Transaction API

JTS Java Transaction Service

JVM Java Virtual Machine

JWS JWorkShop

MBean Managed Bean

MVC Model View Controller

PKI Public Key Infrastructure

PK Primary Key

RA Registration Authority

SQL Structured Query Language

UML Unified Modeling Language

WAP Wireless Application Protocol

WML WAP Wireless Markup Language

XML EXtensible Markup Language

Literaturverzeichnis

[AS02] ANDREAL SCHAEFER, et a. *XDoclet*. http://xdoclet.sourceforge.net/index.hml. 2002

[Com02a] COMPANY, MySQL A. *MySQL*. http://www.mysql.com. 2002

[Com02b] COMPANY, MySQL A. *MySQL Documentation*. http://www.mysql.com/doc/en/index.html. 2002

[Com02c] COMPANY, MySQL A. *MySQL Installation Documentation*. http://www.mysql.com/doc/en/Installing.html. 2002

[EG01] ERICH GAMMA, et a.: *Entwurfsmuster, Elemente wiederverwendbarer objektorientierter Software*. 1. Aufl. Addison-Wesley, 2001. – ISBN 3827318629

[EG02] ERICH GAMMA, Kent B. *JUnit, Testing Resources for Extreme Programming*. http://sourceforge.net/projects/junit. 2002

[ER02a] Kap. 13 EJB Best Practices and Performance Optimizations In: ED ROMAN, Tyler J.: *Mastering Enterprise JavaBeans, Second Edition*. New York : Wiley Computer Publishing, 2002, S. 400–402. – ISBN 0–471–41711–4

[ER02b] Kap. 14 Clustering In: ED ROMAN, Tyler J.: *Mastering Enterprise JavaBeans, Second Edition*. New York : Wiley Computer Publishing, 2002, S. 419–423. – ISBN 0–471–41711–4

[Fel02] FELDGEN, Lutz. *Analyse von Enterprise Application Servern und Datenbank Management Systemen für FlexiTRUST*. Darmstadt, Technische Universität Darmstadt, Fachgebiet Theoretische Informatik Kryptographie und Computeralgebra, Studienarbeit, September 2002

[Fou02] FOUNDATION, Apache S. *Struts*. http://jakarta.apache.org/struts. 2002

[Gro02a] GROUP, ExoLab. *Tyrex*. http://tyrex.exolab.org. 2002

[Gro02b] GROUP, JBoss. *JBoss Enterprise Application Server*. http://www.jboss.org. 2002

[Inc02a] INC., Sun M. *J2EE Application Server*. http://java.sun.com/j2ee/sdk_1.3/. 2002

[Inc02b] INC., Sun M. *The J2EE Tutorial - Enterprise Beans*. http://java.sun.com/j2ee/tutorial/doc/EJBConcepts.html. 2002

[Inc02c] INC., Sun M. *Java 2 Platform, Standard Edition*. http://java.sun.com/j2se/1.4.1/. 2002

[Inc02d] INC., Sun M. *Java Authentication and Authorization Service*. http://java.sun.com/products/jaas/. 2002

[Inc02e] INC., Sun M. *Java Transaction API*. http://java.sun.com/products/jta. 2002

[Inc02f] INC., Sun M. *Java Transaction Service*. http://java.sun.com/products/jts. 2002

[Inc02g] INC., Sun M. *Sun Java Center J2EE Patterns - Data Access Object*. http://developer.java.sun.com/developer/restricted/patterns/DataAccessObject.html. 2002

[Inc02h] INC., Sun M. *Sun Java Center J2EE Patterns - Service Locator*. http://developer.java.sun.com/developer/restricted/patterns/ServiceLocator.html. 2002

[JG02] JBOSS GROUP, Andreas S. *JBoss 3.0 Quick Start Guide*. http://unc.dl.sourceforge.net/sourceforge/jboss/JBoss.3.0QuickStart.pdf. 2002

[Lab02] LABORATORIES, RSA. *PKCS #11 - Crytographic Toke Interface Standard*. http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/. 2002

[Oes01] OESTEREICH, Bernd: *Objektorientierte Softwareentwicklung*. München : Oldenburg Verlag, 2001. – ISBN 3486255738

[Org02] ORG, AspectJ. *AspectJ*. http://aspectj.org. 2002

[Pro02] PROJECT, The J. *Log4j Project*. http://jakarta.apache.org/log4j/docs/index.html. 2002

[Ros02] ROSSMÜLLER, Oliver. *JUnitEE*. http://www.junitee.org. 2002

[SJG02a] Kap. 2 The JBoss JMX Microkernel In: STARK, Scott ; THE JBOSS GROUP: *JBoss Administration and Development Second Edition*. 2520 Sharondale Dr., Atlanta, GA 30305 USA : JBoss Group, LLC, Oktober 2002, S. 39–94

[SJG02b] Kap. 4 Transactions on JBoss - The JTA Transaction Service In: STARK, Scott ; THE JBOSS GROUP: *JBoss Administration and Development Second Edition*. 2520 Sharondale Dr., Atlanta, GA 30305 USA : JBoss Group, LLC, Oktober 2002, S. 133–140

[SLJG02] Kap. 4 HA-JNDI In: SACHA LABOUREY, Bill B. ; THE JBOSS GROUP: *JBoss Clustering*. 2520 Sharondale Dr., Atlanta, GA 30305 USA : JBoss Group, LLC, September 2002, S. 24–29

[Sof02] SOFTWARE, Minq. *DbVisualizer 3.1*. http://www.minq.se/products/dbvis/info.html. 2002

[Sou02] SOURCEFORGE. *EJTools JMX Browser*. http://ejtools.sourceforge.net. 2002

[Tog02] TOGETHERSOFT. *Together ControlCenter 6.0*. http://www.togethersoft.com/products/controlcenter. 2002

[Wie01] WIESMAIER, Alexander. *FlexiTRUST CA - Ein flexibles, skalierbares und dynamisch erweiterbares Trustcenter*. Darmstadt, Technische Universität Darmstadt, Fachgebiet Theoretische Informatik Kryptographie und Computeralgebra, Diplomarbeit, Februar 2001

[Win02] WINKLER, Markus. *Analyse der Enterprise-Java-Beans Spezifikation Version 2.0 als Basis für FlexiTRUST*. Darmstadt, Technische Universität Darmstadt, Fachgebiet Theoretische Informatik Kryptographie und Computeralgebra, Studienarbeit, September 2002